Oracle® Database

Using Oracle Blockchain Platform

Release 21.1.2
F20801-05
April 2021

Oracle Database Using Oracle Blockchain Platform, Release 21.1.2

F20801-05

Primary Author: Oracle Blockchain Platform development, product management, and quality assurance teams

Contents

Preface

Administering Oracle Blockchain Platform explains how to provision and maintain Oracle Blockchain Platform instances.

Topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

This guide is intended for administrators responsible for using and managing Oracle Blockchain Platform blockchains .

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see these Oracle resources:

- *Administering Oracle Blockchain Platform*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

What's Oracle Blockchain Platform?

This topic contains information to help you understand what Oracle Blockchain Platform Enterprise Edition is.

Topics:

What's a Blockchain?

A blockchain is a system for maintaining distributed ledgers of facts and the history of the ledger’s updates. A blockchain is a continuously growing list of records, called blocks, that are linked and secured using cryptography.

This allows organizations that don't fully trust each other to agree on the updates submitted to a shared ledger by using peer to peer protocols rather than a central third party or manual offline reconciliation process. Blockchain enables real-time transactions and securely shares tamper-proof data across a trusted business network.

A blockchain network has a founder that creates and maintains the network, and participants that join the network. All organizations included in the network are called members.

Oracle Blockchain Platform is a permissioned blockchain, which provides a closed ecosystem where only invited organizations (or participants) can join the network and keep a copy of the ledger. Permissioned blockchains use an access control layer to enforce which organizations have access to the network. The founding organization, or blockchain network owner, determines the participants that can join the network. All nodes in the network are known and use consensus protocol to ensure that the next block is the only version of truth. There are three steps to consensus protocol:

- **Endorsement —** This step determines whether to accept or reject a transaction.
- **Ordering —** This step sorts all transactions within a time period into a sequence or block.
- **Validation —** This step verifies that the required endorsement are gotten in compliance with the endorsement policy and organization permissions.

Blockchain's key properties

Shared, transparent, and decentralized— The network maintains a distributed ledger of facts and update history. All network participants see consistent data. Data is distributed and replicated across the network’s organizations. Any authorized organizations can access data.

Immutable and irreversible — Each new block contains a reference to the previous block, which creates a chain of data. Data is distributed among the network organizations. Blockchain records can only be appended and can't be undetectably altered or deleted. Consensus is required before blocks or transactions are written to the ledger. Therefore, the existence and validity of a data record can't be denied. After endorsement policies are satisfied and consensus is reached, data is grouped into blocks and blocks are appended to the ledger with cryptographically secured hashes that provide immutability. Only those members authorized to have the corresponding encryption keys can view data.

Encryption — All records are encrypted.

Closed ecosystem — Joined organizations can have a copy of the ledger. Organizations are known in the real world. Consensus protocols depend on knowing who the organizations are.

Speed — Transactions are verified in minutes. Network members interact directly.

Blockchain example

An example of an organization that benefits from using blockchain is a supply chain contract manufacturing company. Suppose this company is located in the United States and uses a third-party company in Mexico to source materials for and produce electronic components. With a blockchain network, the manufacturing company can quickly know the answers to the following questions:

- Where is the product in the production cycle?
- Where is the product being produced?
- Does the product contain ethically sourced materials?
- Does the product meet specifications and exporting compliance rules?
- When is ownership transferred?
- Does the invoice match and should the organization pay it?
- How should the organization handle any exceptions to the manufacturing, shipping, or receiving process?

Why Should I Use Blockchain?

Implementing blockchain can help you manage and bring efficiency to many aspects of your business practices.

The key benefits of using a blockchain are:

Increase Business Velocity — You can create a trusted network for business-to-business transactions and extend and automate your operations beyond the enterprise. With blockchain, you can optimize business decisions by providing real-time information visibility across your company's ecosystem.

Reduce Operation Costs — Use blockchain to accelerate transactions and eliminate cumbersome offline reconciliations by using a trusted shared fabric of common information. Blockchains help you eliminate intermediaries and related costs, possible single points of failure, and time delay by using a peer to peer business network.

Reduce the cost of fraud and regulatory compliance — Blockchain allows you to gain the security of knowing that business critical records are made tamper-proof with securely replicated, cryptographically linked blocks that protect against single point of failure and insider tampering.

What Are the Advantages of Oracle Blockchain Platform?

Using Oracle Blockchain Platform to create and manage your blockchain network has many advantages over other available blockchain products.

As a preassembled platform, Oracle Blockchain Platform includes all the dependencies required to support a blockchain network: compute, storage, containers, identity services, event services, and management services. Oracle Blockchain Platform includes the blockchain network console to support integrated operations. This helps you start developing applications within minutes, and enables you to complete a proof of concept in days or weeks rather than months.

How Oracle Blockchain Platform Adds Value to Hyperledger Fabric

Oracle Blockchain Platform is based on the Hyperledger Fabric project from the Linux Foundation, and it extends the open source version of Hyperledger Fabric in many ways.

Enhances Security

- Uses data in-transit encryption based on TLS 1.2, prioritizing forward-secrecy ciphers in the TLS cipher-suite.
- Uses data at-rest encryption for all configuration and ledger data.
- Provides audit logging of all API calls to the blockchain resources, with records available through an authenticated, filterable query API.

Adds REST Proxy

- Supports a rich set of Fabric APIs through REST calls for simpler transaction integration. See REST API for Oracle Blockchain Platform.
- Enables synchronous and asynchronous invocations. Enables events and callbacks and DevOps operations.
- Simplifies integration and insulates applications from underlying changes in transaction flow.

Provides the Management and Operations Console

- Provides a comprehensive, intuitive web user interface and wizards to automate many administration tasks. For example, adding organizations to the network, adding new nodes, creating new channels, deploying and instantiating chaincodes, browsing the ledger, and more.
- Enables DevOps through REST APIs for administration and monitoring of blockchain.
- Dynamically handles configuration updates without node restart.
- Includes dashboards, ledger browser, and log viewers for monitoring and troubleshooting.

Replaces Ledger DB World State Store With Oracle Berkeley DB

- Provides Couch DB rich query support at Level DB performance.

- Provides SQL-based rich query support. See What's the State Database?
- Validates query results at commit time to ensure ledger integrity and avoid phantom reads.

Integrates Rich History Database

- Enables transparent shadowing of transaction history to Autonomous Data Warehouse or Database as a Service and the use of Analytics or Business Intelligence (for example, Oracle Analytics Cloud or third-party tools) on blockchain transaction history and world state data. See Create the Rich History Database.
- Supports standard tables and blockchain tables for storing rich history. Blockchain tables are tamperproof append-only tables, which can be used as a secure ledger while also being available for transactions and queries with other tables.

Highly Available Architecture and Resilient Infrastructure

Built for business-critical enterprise applications, Oracle Blockchain Platform is designed for continuous operation as a highly secure, resilient, scalable platform. This platform provides continuous monitoring and autonomous recovery of all network components based on continuous backup of the ledger blocks and configuration information.

Each customer instance uses a framework of multiple managed VMs and containers to ensure high availability. This framework includes:

- Peer node containers distributed across multiple VMs to ensure resiliency if one of the VMs is unavailable or is being patched.
- Orderers, fabric-ca, console, and REST proxy nodes are replicated in all VMs for transparent takeover to avoid outages.
- A highly available Raft/Zookeeper cluster, leveraging multi-VM deployment topology.
- Isolated VM environments for customer chaincode execution containers for greater security and stability.

What Do I Get with Oracle Blockchain Platform?

Your instance includes everything you need to build, run, and monitor a complete production-ready blockchain network based on Hyperledger Fabric.

Your Oracle Blockchain Platform instance is defined by the options you selected when you created your instance. Your instance includes validating peer nodes, a membership services provider (MSP), and an ordering service.

In addition, REST proxy nodes are provided and a default channel is created. Use the console user interface to further configure, administer, and monitor the network, as well as install, instantiate, and upgrade smart contracts (also known as chaincodes). The Developer Tools tab contains sample chaincodes that you can deploy and run to help you quickly understand how the blockchain network works.

Oracle Blockchain Platform is pre-assembled with underlying services, including containers, compute, storage, LDAP for authentication, object store for embedded archiving, and management and log analytics for operations and troubleshooting. You can configure multiple peer nodes and channels for availability, scalability, and confidentiality, and Oracle Blockchain Platform will automatically handle the underlying dependencies.

2

Get Started Using Samples

This topic contains information about the samples included in your instance. Using samples is the fastest way for you to get familiar with Oracle Blockchain Platform.

Topics

What Are Chaincode Samples?

Oracle Blockchain Platform includes chaincode samples written in Go and Node.js to help you learn how to implement and manage your network's chaincodes.

To get to the Chaincode Samples page in the Oracle Blockchain Platform console, open the **Developer Tools** tab and select **Samples**.

The Chaincode Samples page contains:

- The Balance Transfer sample is a simple chaincode representing two parties with account balances and operations to query the balances and transfer funds between parties.
- The Marbles sample includes a chaincode to create marbles where each marble has a color and size attribute. You can assign a marble to an owner and enable operations to query status and trade marbles by name or color between owners.
- The Car Dealer sample includes a chaincode to manage the production, transfer, and querying of vehicle parts; the vehicles assembled from these parts; and transfer of the vehicles.

  In this sample, a large auto maker and its dealers and buyers have created a blockchain network to streamline its supply chain activities. Blockchain helps them reduce the time required to reconcile issues with the vehicle and parts audit trail.

Use the **Download sample here** links under each sample to download the sample chaincode. The download contains the Go and Node.js versions of the chaincode.

The download also contains a Java version of the chaincode.

For information about installing, instantiating, and invoking the samples on your instance, see Explore Oracle Blockchain Platform Using Samples.

Explore Oracle Blockchain Platform Using Samples

You can install, instantiate, and invoke the sample chaincodes included in Oracle Blockchain Platform.

Tutorial

You must be an administrator to install and instantiate sample chaincodes. If you've got user permissions, then you can invoke sample chaincodes.

1. Go to the console and select the **Developer Tools** tab.
2. Click the **Samples** pane.

   The Blockchain Samples page is displayed.
3. Locate the sample chaincode and install it.
   a. Choose the sample chaincode that you want to use and click the corresponding **Install** button.
   b. In the Install Chaincode dialog, specify one or more peers to install the chaincode on, and select which chaincode language you want to use (Go, Node.js, or Java). Click **Install**.
4. Instantiate the chaincode.
   a. Click the chaincode’s **Instantiate** button.
   b. In the Instantiate Chaincode dialog select the channel you want to instantiate the chaincode to, and specify any required parameters. Click **Instantiate and Enable in REST Proxy**.
5. Go to the Channels tab and click the name of the channel that you instantiated the sample chaincode to.
   a. In the Channel Information page, click the Instantiated Chaincodes pane to confirm the chaincode's deployment on the channel.
   b. You can use the Ledger area to locate information about individual transactions on the channel.
6. Click the Ledger pane and confirm the following.
   - The Ledger Summary indicates one deployment occurred.
   - In the Ledger table, locate the block with the Type of **data (sys)**.
   - Click the block and in the Transactions table, click the arrow icon to display more information about the block. Confirm that the **Function Name** field displays “deploy.”
7. If needed, go to the Chaincodes tab and instantiate the chaincode on other channels.

   If you're working on a network that contains multiple members and have instantiated the chaincode on the founder, then you don’t have to instantiate the chaincode on the participants where you installed the same chaincode. In such cases, the chaincode is already instantiated and running on the participants.
   a. Locate the name of the chaincode you want to instantiate in the table and click it.
   b. In the Chaincode Information page, click the **Instantiate on a New Chaincode** button.
   c. In the Instantiate Chaincode dialog specify the required information.
8. Invoke the chaincode.
   a. Go to the Blockchain Samples page, locate the chaincode you're working with, and click its **Invoke** button.
   b. In the Invoke Chaincode dialog, select a channel to run the transaction on.

   c. In the **Action** field, specify an action to execute the chaincode.
   d. Click **Execute**. The Transaction Results shows returned values, and the API details field displays the detailed log of all blockchain processes performed from invoking the transaction.

9. Confirm whether the chaincode invoked successfully.
   a. Go to the Channels tab, and locate and click the channel the chaincode was installed on.
   b. Confirm that the Ledger pane is selected, and in the Query Ledger table, locate the block number indicating that an invocation occurred.
   c. Click the block and confirm that in the Transactions table you see “Success” in the **Status** column.

10. If needed, go to the Samples page and invoke any other operations on the chaincode.

3

Manage the Organization and Network

This topic contains information to help you understand the console and how you can use it to manage the channels and nodes that make up your organization and the blockchain network.

Topics

What's the Console?

The Oracle Blockchain Platform console helps you monitor the blockchain network and perform day to day administrative tasks.

When you provisioned your Oracle Blockchain Platform instance, all of the capabilities you need to begin work on your blockchain network were added to the console.

You can use the console to perform tasks such as managing nodes, configuring network channels and policies, and deploying and instantiating chaincodes. You can also monitor and troubleshoot the network, view node status, view ledger blocks, and find and view log files.

In most cases, each member of your network has its own console that they use to manage their organization and monitor the blockchain network. Your role in the network (founder or participant) determines the tasks you can perform in your console. For example, if you're a participant, then you can’t add another participant to the network. Only the founder can add a participant to the network.

Also, what you can do in the console is determined by your access privileges (either Administrator or User). For example, only an Administrator can set an anchor peer or create a new channel.

Your instance includes sample chaincodes that you can use to get started. See Explore Oracle Blockchain Platform Using Samples.

The console is divided into tabs.

Dashboard Tab

Use the Dashboard tab for an overview of the network's performance. See What Type of Information Is on the Dashboard?

On the Dashboard tab, you'll find:

- A banner showing you how many different components are on your network. For example, how many channels and chaincodes.
- The number of user transactions on a channel during a specific time range.
- The number of nodes that are running or stopped.
- The number of endorsements and commits by peers.
- Utilization statistics for your instance's partitions.

Network Tab

The Network tab is where you view a list of the members in your network. The first time you use the Network tab after setting up your instance, you'll see the nodes you created during set up.

You can use the Network tab to:

- Find the organization IDs of the members in your network, their Membership Service Provider (MSP) IDs, and roles.
- Add a participant to the network.
- See a graphical representation of the network's structure.
- Configure, view, or import the orderer settings.
- Manage certificates.
- Add new orderering service node into network.
- Export the network config block.

Nodes Tab

Go to the Nodes tab to view a list of the nodes in your network. The first time you use the Nodes tab after setting up your instance, you'll see:

- The console node.
- The number of peer nodes you requested when provisioning.
- The number of orderer nodes associated with your instance type. Standard has three orderer nodes and cannot be scaled up, while Enterprise has three and additional can be added.
- One Fabric certificate authority (CA) node representing the membership service.
- One REST proxy node.

During the founder instance provisioning a default channel was created and all peers were added to it.

Use the Nodes tab to:

- View and set node configurations.
- Export and import peers.

- Start, stop, and restart nodes.
- Configure and start a new orderer node.
- See a graphical representation of which peer nodes are using which channels.
- Click a node's name to find more information about it.

Channels Tab

The Channels tab shows you the channels in your network, the peers using the channels, and the chaincodes instantiated on the channels. The first time you use the Channels tab after setting up your instance, you’ll see the default channel that was created and all of the peers in your network added to it.

Use the Channels tab to:

- Add new channels.
- See the number of chaincodes instantiated on a channel.
- Click a channel's name to find more information about it, such as its ledger summary, the peers and OSNs joined to the channel, and the channel's policies and ACLs.
- Join peers to the channel.
- Manage the ordering service of the channel.
- Add or remove an ordering service node (OSN) for a channel.
- View and update the ordering service’s settings.
- Configure rich history for the channel.

Chaincodes Tab

Note that Oracle Blockchain Platform refers to smart contracts as chaincodes.

Go to the Chaincodes tab to view a list of the chaincodes installed on the instance. The first time you use the Chaincodes tab after setting up your instance, no chaincodes display in the list because no chaincodes were included during set up. You must add the needed chaincodes.

You can use the Chaincodes tab to:

- Install, instantiate, and deploy a chaincode using the Quick or Advanced deploy option.
- See how many peers have a chaincode installed.
- Find out how many channels a chaincode was instantiated on.
- Upgrade a chaincode.
- Find the chaincode’s path.

Developer Tools Tab

The Developer Tools tab is designed to help you learn blockchain fundamentals like how to write chaincodes and create blockchain applications.

You can use the Developer Tools tab to:

- Find templates and the Hyperledger Fabric mock shim to help you create chaincodes.
- Link to the SDKs and APIs that you need to write blockchain applications.
- Use the sample chaincodes to learn about chaincodes. Install, instantiate, and invoke the sample chaincodes.
- Download the Blockchain App Builder for Oracle Blockchain Platform - a set of tools and samples to help you create, test, and debug chaincode projects using a command line interface or a Visual Studio Code extension.

Modify the Console Timeout Setting

The Oracle Blockchain Platform console attempts to contact the nodes on the network for 600 seconds before it times out.

In most cases you won’t have to adjust this setting, but if the console is frequently not responding, then consider increasing the timeout value. Oracle doesn’t recommend decreasing the timeout value.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab go to the nodes table and locate the console node. Use the nodes table’s type column to find the Console node.
3. Click the node’s **More Actions** menu and then click **Edit Configuration**.

   The Configure dialog is displayed.
4. In the **Request Timeout (s)** field, type or use the arrow buttons to indicate the timeout length in seconds.
5. Click **Submit**.

   The timeout length changes immediately, and you don’t have to restart the console.

Find and Understand Your Oracle Blockchain Platform Version Number

Use this topic to find and understand your Oracle Blockchain Platform instance's version number.

1. Go to the console and in the top right of the screen, locate and click your user name.
2. Select **About**.

   Your instance's version number will look similar to:

   ```
   20.3.1
   ```

   where:

   - 20 is year
   - 3 is the quarter
   - 1 is the minor release number

Monitor the Network

This topic contains general information about monitoring the blockchain network.

Topics:

- How Can I Monitor the Blockchain Network?
- What Type of Information Is on the Dashboard?
- View Network Activity

How Can I Monitor the Blockchain Network?

The console provides several ways for you to monitor the activity and health of your blockchain network.

For example, you can find summary information about the total number of blocks submitted to the ledger, or you can search for and locate information about specific chaincode transactions that happened on a specific channel.

You can use the console to locate the following sources of information to help you understand what's happening on your network.

Network Overview Information

Use the Dashboard tab if you need at-a-glance information about how well the whole network is working and to spot any general issues such as a high rate of failing transactions. See View Network Activity .

Ledger Summary

For information about the runtime statistics for transactions on a specific channel, go to the channel's Ledger Summary area. You can drill into a specific transaction for more information about it, such as which member initiated the transaction and which peer endorsed it. See View a Channel's Ledger Activity.

Node Health

Use a node's Health Summary area to help you understand how the node is performing on the network. For example, CPU utilization and memory utilization. See:

- View Health Information for a CA Node
- View Health Information for the Console Node
- View Health Information for an Orderer Node
- View Health Information for a Peer Node
- View Health Information for a REST Proxy Node

What Type of Information Is on the Dashboard?

The console’s Dashboard tab provides an overview of how well your network is functioning. You can use this information to identify any issue and to navigate to other tabs in the console where you can learn more about and resolve any issues.

Summary Bar

This section shows the components in your network (for example, how many nodes and chaincodes). You can click a component number to go to the console tab for more information or to perform tasks related to the component. If your instance is a development instance, then “Development mode” is displayed in the bottom right of the summary bar.

At the top of the console, you’ll see what type of instance you’re working with. If you’re a network founder, then you’ll see “(Founder)”. If you’re a participant in a network, then the top of your console displays the name of the network you’re joined to. For example, “(Participant of <*foundername*>)”.

Health

This section shows how many nodes are running and how many are stopped in the network. Click the node numbers to go to the Nodes tab to investigate why a node might be stopped, or for more information about the nodes in the network.

The nodes in your network are partitioned inside of a virtual machine (VM). This section also shows the percentage of the partition memory used, and the percentage of CPU and disk used. If the memory percentage is relatively low (for example, 50% or lower), then you can create another peer node without your system’s performance decreasing significantly. If the percentage is close to 100, then your system most likely can’t support another peer node.

Channel Activity

This area shows how many blocks have been created and how many transactions have been executed based on the number of blocks created. Note that you might see more blocks created than user transactions. For example, if you create a new channel or you instantiate a chaincode, then those are classified as system-level transactions and are included in blocks, but not classified as user transactions. This area shows the top four channels that have handled the most transactions, and for each channel shows the number of transactions that have succeeded and failed.

Note the following information:

- User transactions are transactions that were invoked as part of the chaincode’s execution, and not underlying actions such as setting up the network, creating channels, and installing and instantiating chaincodes.
- A block can contain multiple user transactions.

You can filter the amount of activity information that is displayed. You can select a set time range (for example, last hour or last week), or you can select **Custom** and pick the dates you want activity information for.

Peer Activity

This area shows the number of endorsement and commits completed by the network’s peer nodes. This area shows the top four peer nodes that have endorsed and committed the most transactions, and for each of those four peers, this area shows the number of endorsements and commits that have succeeded and failed.

Note the following information:

- A transaction is an endorsement, and a commit is when a transaction is written to the block.
- Commits can be either user transactions or system transactions
- Commits are the number of transactions that have been committed to the block. Commits aren’t blocks.
- Only specific peers must do endorsements, but all peers must do commits.

You can filter the amount of activity information that is displayed. You can select a set time range (for example, last hour or last week), or you can select **Custom** and pick the dates you want activity information for.

View Network Activity

Use the console’s Dashboard tab to find information about your blockchain network’s activities, such as percentage of nodes that are running or stopped, and how successfully the network is executing chaincode transactions.

You can use this information as a starting place and then use the other tabs in the console to drill into any issues that you discover. For information about what displays in the Dashboard tab, see What Type of Information Is on the Dashboard?

1. Go to the console and select the Dashboard tab.
2. To see channel and peer activity information that occurred at a specific time such as for the last week or month, go to the filter dropdown menu and select the time range you want. Select Custom to enter specific begin and end dates and click **Apply**.

Manage Nodes

This topic contains general information about managing the nodes in your network.

Topics

- What Types of Nodes Are in a Network?
- Find Information About Nodes
- Start and Stop Nodes
- Restart a Node
- Set the Log Level for a Node

What Types of Nodes Are in a Network?

A blockchain network contain console, peer, orderer, certification authority (CA), and REST proxy nodes. The nodes that display in your console depend upon if you're the founder of or a participant in a network.

For example, if you're a participant in a network, your console won't display an orderer node for that network. If you're a founder, your console displays all node types.

What nodes are included in a new instance?

After you provision your instance and access the Nodes tab for the first time, you'll see:

- One console node.
- The number of peers you requested during set up. These peers display with the Peer(Member) type. The maximum number of peer nodes that can be included with an instance is 16.
- An orderer node representing an ordering service. Starting in version 20.3.1, both founder and participant instances can have ordering service nodes (OSNs).
- A Fabric certificate authority (CA) representing the membership service.
- A REST proxy node.

I need more information about the different node types

Use this table to find more information about nodes.

Node Type	What Does This Node Do?	Displays In Founder or Participant Instance	Number of Nodes per Instance	Can I Add Another Node After Provisioning My Instance?
CA	This node provides and manages peer node credentials and member credentials.	Founder Participant	1	No
Console	This node is the console component.	Founder Participant	1	No
Orderer	This node provides communication between nodes. It guarantees the delivery of transactions into blocks and blocks into the blockchain. If you're a participant, then you must import the founder's ordering service setting into your instance so that all peer nodes can communicate.	Founder Participant	3	Enterprise Edition: Yes Standard Edition: No

Node Type	What Does This Node Do?	Displays In Founder or Participant Instance	Number of Nodes per Instance	Can I Add Another Node After Provisioning My Instance?
Peer	This node contains a copy of the ledger and writes transactions to the ledger. This node can also endorse transactions. Your network can contain member or remote peers.	Founder Participant	2 to 16 The number of peer nodes you can add was specified when your instance was created.	Yes
REST Proxy	This node maps an application identity to a blockchain member, which allows users and applications to call the Oracle Blockchain Platform REST APIs.	Founder Participant	1	No

Find Information About Nodes

This topic contains information about where in the console you can find information about the nodes in your instance and network.

Topics:

- View General Information About Nodes
- Access Information About a Specific Node
- View a Diagram of the Peers and Channels in the Network
- Find Node Configuration Settings

View General Information About Nodes

Use the Nodes tab to view general information about all of the nodes in your network. For example, Name, Route, Type, and Status.

You can also use the Nodes tab to drill into details about a specific node. For more information about node types, see What Types of Nodes Are in a Network?

1. Go to the console and select the Nodes tab.
2. In the Nodes tab confirm that the **List View** (and not the **Topology View**) is displaying.

Column	Description
Route	Oracle Blockchain Platform generated the URLs when you provisioned your instance or when you create new nodes. If you use the Hyperledger Fabric SDK, then you need these URLs to specify which peers you want the SDK to interact with.
Type	Indicates the node type.
MSP ID	Membership Service Provider ID.

Column	Description
Status	Indicates if the node is running or down. Also indicates if there's an unapplied configuration change for the node. Note the following statuses: • **Up** — The node is running and working normally. • **Down** — The node is stopped. • **N/A** — This status displays for remote peers because your instance doesn't have the permissions required to get the peer's status.
IsConfigured	If the node's configuration was updated you need to restart the node for the updates to take effect. Nodes with the `yes` status are running (and not stopped).
More Actions Menu	Your permissions determine the options available from the More Actions menu. If you're an administrator, this button provides links to modify the node's configuration. Administrators and users can stop, start, and restart nodes.

Access Information About a Specific Node

Use the Nodes tab to access information about a specific. For example, health information or log files.

1. Go to the console and select the Nodes tab.
2. Click a node's name to go to the Node Information page. The panes that display in the Node Information page depend on the node type you select.

Pane	Available for Which Node Types?	What can I do in this pane?
Health	All	View metrics to help you understand how the node is performing on the network. Example of metrics include CPU Utilization and Memory Utilization. For a Peer node, this pane displays information about endorsed and committed transactions.
Channels	Peer	View a list of channels the selected peer node is using for its communications with other nodes. Join the peer node to other existing channels as needed. Go to the Channel page to create a new channel and specify which peer nodes can join it.
Chaincodes	Peer	View the chaincodes that are installed on the peer node. Go to the Chaincode page to install a new chaincode or upgrade an existing chaincode.
Transaction Statistics	REST proxy	View the total queries, failed queries, total invocations, and failed invocations handled by the REST proxy.

View a Diagram of the Peers and Channels in the Network

Use the Topology view to access an interactive diagram that shows which network peers are using which channels.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click **Topology View** to see a diagram showing the peer nodes in your network and which channels they're using.
3. Hover over a peer to highlight it and the channels it's using.

Find Node Configuration Settings

Use the Nodes tab to find a specific node's configuration settings. If you're an administrator, then you can update a node's configuration settings. If you're a user, then you can view a node's configuration settings.

1. Go to the console and select the Nodes tab.
2. Go to the Nodes table, locate the node that you want configuration setting information for, and click the node's **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed, showing the attributes specific to the node type you selected. See Node Configuration.

Start and Stop Nodes

You can start or stop CA, orderer, peer, and the REST proxy nodes in your network. You can't start or stop the console node or remote peer nodes.

You can start and stop nodes depending upon the traffic in your network. For example, if network traffic is light, then you can stop unneeded peer nodes and orderer nodes.

You can also restart a node. See Restart a Node.

When you stop a peer node, Oracle Blockchain Platform removes the peer's listing on the Channel tab and Chaincodes tab. If you stop all peers that have the chaincode installed, then the Chaincodes tab doesn't list the chaincode. If you stop all peers joined to a channel, then the Channels tab lists the channel, but its information isn't available to view.
Before stopping a node for an extended period of time, you should transfer all this peer's responsibilities to other running peers, and then remove all the responsibilities this peer has.

- Check all other peers' gossip bootstrap address lists, remove the peer address, and add another running peer's address if needed. After peer configuration change, restart the peer.
- Check all channels' anchor peer lists, remove the peer from the anchor peer lists, and add another running peer to the anchor peer list if needed.
- If a channel or chaincode is only joined or instantiated in this peer, you should consider using another running peer to join the same channel and instantiate the same chaincode.

You must be an administrator to perform this task.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the node that you want to start or stop, and click the node's **More Actions** button.
3. Click either the **Start** or **Stop** option. The node's status changes to either *up* or *down* and information is written to the node's log file.

Restart a Node

You can restart the CA, orderer, peer, and REST proxy nodes in your network. You can't restart the console node or remote peer nodes.

You should restart a node if it's not responding or running properly, or if you've updated a node's configuration. You can also start or stop a node. See Start and Stop Nodes.

You must be an administrator to perform this task.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the node that you want to restart, and click the node's **More Actions** button.
3. Click **Restart**.

   The node's status changes to *restarting* and information is written to the log file.

Set the Log Level for a Node

If you're an administrator, then you can specify the type of information you want to include in a node's log files. For example, ERROR, WARNING, INFO, or DEBUG.

By default, every node's log level is set to INFO. When developing and testing your network, Oracle suggests that you set the logging level to DEBUG. If you're working in a production environment, then use ERROR.

Only an administrator can change a node's log level setting. If you're a user, then you can view a node's log level settings.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the nodes table, locate the node you want to update, click its **More Actions** menu, and click **Edit Configuration**.

   If you have user permissions, then your console will have the **View** option that you click to see the node's log level setting and other configuration settings.

   The Configure dialog is displayed.
3. In the **Log Level** field, select the log level you want to use.
4. Click **Submit**.

Manage Channels

This topic contains information about managing the channels in your network.

Topics

- Add an Anchor Peer
- Change or Remove an Anchor Peer
- View Information About Instantiated Chaincodes
- Work With Channel Policies and ACLs
- Add or Remove Orderers To or From a Channel
- Set the Orderer Administrator Organization
- Edit Ordering Service Settings for a Channel

What Are Channels?

Channels partition and isolate peers and ledger data to provide private and confidential transactions on the blockchain network.

Members define and structure channels to allow specific peers to conduct private and confidential transactions that other members on the same blockchain network can't see or access. Each channel includes:

- Peers
- Shared ledger
- Chaincodes instantiated on the channel
- One or more ordering service nodes
- Channel policy definitions and ACLs where the definitions are applied

Each peer that joins a channel has its own identity that authenticates it to the channel peers and services. Although peers can belong to multiple channels, the information on transactions, ledger state, and channel membership is restricted to peers within each channel.

You can use the Oracle Blockchain Platform console or the Hyperledger Fabric SDK to create channels on your blockchain network. See View Channels.

View Channels

Members in your network use channels to privately communicate blockchain transactions information.

Use the Channel tab to view a list of the channels in your network, create and monitor channels, specify anchor peers, and upgrade the instantiated chaincodes used on your channels.

1. Go to the console and select the Channels tab.

   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.

2. In the channel table, click the channel name you want information about. Note that if all peers joined to the channel are stopped, then the channel is listed but its information isn't available to view.

   The Channel Information page is displayed.

3. Click through the Channel Information page's panes to find information about the channel.

Section	What can I do in this pane?
Ledger	Get information about the channel’s ledger activity such as block number and the number of user transactions in the block. Click a block number to drill into information about its transactions. You can use the filter field to specify the summary information that you want to see (for example, information from the last day or last month), or use the custom option to enter start and end times. See View a Channel’s Ledger Activity.
Instantiated Chaincodes	View the list of chaincodes that have been instantiated on the channel.
Orderers	View a list of the orderers currently active, and allows you to add a new OSN to join the channel.
Peers	View the list of peers that are joined to the channel. Use this section to set anchor peers for the channel.
Organizations	View the list of network members whose peers are using the channel to communicate.
Channel Policies	View the list of the standard policies and any policies that you created for the channel. Use this section to add, modify, and delete policies.
ACLs	View the access control lists (ACLs) and the policies used to manage which organizations and roles can access the channel's resources.

Create a Channel

You can add channels to the network and specify which members can use the channel, and which peers can join the channel. You can’t delete channels.

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.
2. In the Channels tab, click **Create a New Channel**.
3. In the **Channel Name** field, enter a unique name for the channel. The channel's name can be up to 128 characters long.
4. In the **Application Capabilities** field, select `1_4_2` as the capabilities level for the channel. Don't select `1_1` or `1_3` - they were used in earlier versions of the product and selecting them will remove support for newer product features.
5. In the **Creator Organization** field, select the parent or child organization that you want to administer the channel.
6. In the Organizations section, select any additional members that you want to communicate on the channel.

   If you’re working in a participant instance, you need to add the founder to your instance before the founder’s MSP ID displays in the Organization section. To add the founder organization, go to the Network tab and click the **Add Organization** button to upload the founder’s certificates.
7. In the **MSP ID ACL** section, specify the organizations that have access to the channel and permissions for each selected organization. Note that you can add more organizations to or delete them from the channel later, as needed.

   Your organization’s permissions are set to write (ReaderWriter) and you can't modify this setting. By default, other member’s permissions are set to write (ReaderWriter), but you can change them to read (ReaderOnly) if you don’t want the members to invoke chaincodes and to only read channel information and blocks on the channel.

8. (Optional) In the **Peers to Join Channel** field, select one or more peers. Note the following information:
    - Your instance has two VMs (Partition 1 and Partition 2) and Oracle recommends that you join one peer from each partition to the channel. This is because if one VM is unavailable that the channel can still process endorsements and commits. A peer's name tells you which partition it's located in. For example, peer0–1 and peer1–1 are located in Partition 1. And peer0–2 and peer1–2 are located in Partition 2.
    - You can join a maximum of seven peers from Partition 1 and seven peers from Partition 2.
    - If your network contains participants, the participants' peers don't display in this list. Participants must use their consoles to join peers to the channel. A participant can't join its peers to the channel unless its organization was added to the channel's MSP ID ACL section.
    - If you want to create the channel only, then don't select any peers. You can add peers to the channel later.
9. Click **Submit**.

    The channel table displays the new channel.

After you create the channel, you can:

- Instantiate a chaincode on the channel. See Instantiate a Chaincode.
- If the network contains participants, then they use their consoles to join member peers to the channel. See Join a Peer to a Channel.

View a Channel's Ledger Activity

Use the ledger to find summary information and runtime statistics for transactions on a specific channel.

1. Go to the console and select the Channels tab.
2. In the channel table, click the channel name that you want transaction information about. In the Channel Information page, confirm that the Ledger pane is selected.
3. Use the Ledger Summary area to find at a glance information about the channel's activity, such as the total number of blocks in the ledger's chain and the total number of user transactions on the channel.
4. To see blockchain activity that occurred at a specific time such as for the last day or week, go to the filter dropdown menu to select the time range that you want. To locate and drill into a specific set of transactions, select **Custom** and enter search criteria in the **Start Time** and **End Time** fields, or click the calendar icon and pick the dates that you want. Click **Apply**.

    If you select a specific time period (for example, **Last day**) and then select it again to re-run the query, the query doesn't re-run. To get the latest information, click the **Refresh** button.

    Note the following transaction types that can display for a block:

    - genesis — The transaction that runs the configuration block to initialize the channel.

- data (sys) — The transaction that starts the chaincode's container to make the chaincode available for use.
- data — A chaincode transaction called for execution on the channel.

5. To find more information about a specific transaction, locate the transaction in the query ledger table and click it. The transactions table displays the transaction's details.

Transaction Detail	Description
TxID	The unique alphanumeric ID assigned to the transaction. The TxID is constructed as a hash of a nonce concatenated with the signing identity's serialized bytes.
Time	The transaction's time stamp (date and time that the transaction occurred).
Chaincode	Displays the name of the chaincode that executed the transaction. This field can show the name of a chaincode that you wrote, installed, and instantiated, but can also show a system chaincode. System chaincode options are: • LSCC — For lifecycle requests, such as instantiate, install, and upgrade. • QSCC — For querying. This chaincode includes APIs for ledger query.
Status	Shows if the transaction succeeded or failed.

6. Click the triangle next to the TxID to view in depth information about the transaction, such as function name, arguments, validation results, response status, the initiator and the endorser.

   Note that if a transaction failed, then you can use the TxID to search error logs in the peer node or orderer nodes for more information.

View or Update a Channel's Organizations List

You can view the list of the organizations that have access to the channel. If you created the channel, then you can change an organization's permissions on the channel, and you can add organizations to or remove them from the channel

1. Go to the console and select the Channels tab.

   The Channels tab is displayed and the channel table contains a list of all of the channels in your network.

2. In the channels table, locate the channel that you want information about, click the channels **More Actions** button, and click **Edit Channel Organizations**.

   The Edit Organizations page is displayed.

3. In the **MSP ID ACL** section, you can do the following:
   - Modify an organization's permissions. The organization that created the channel is set to write (ReaderWriter). You can't change this setting.
   - If you're the network founder, then clear an organization's checkbox to delete it from the channel. If you're a network participant, then use the **Delete** button to delete an organization from the channel. If you delete an organization from a channel, then the organization and its peers can no longer query, invoke, and

  instantiate a chaincode on the channel. And the removed organization’s peers can’t join the channel.
  - Click an organization’s checkbox to add the organization to the channel and set its permissions. By default, each member’s permissions is set to write (ReaderWriter), but you can change it to read (ReaderOnly) if you don’t want the member to invoke chaincodes and to only read channel information and blocks on the channel.
4. Click **Submit** to save the changes.

Join a Peer to a Channel

You can add a peer node to a channel so that the node can use it to exchange private transaction information with other peer nodes on the channel.

Note the following information:

- When you create a channel, you specify which local peer nodes can join the channel.
- If you’re creating a network containing a participant, then you can select the participant as a member on the channel. Or you can add the participant after the channel is created.
- Your instance has multiple availability domains or fault domains, and Oracle recommends that you join one peer from each partition to the channel. This is because if one VM is unavailable that the channel is still available for endorsements and commits. To determine which domain a peer is located in, in the **More Actions** menu select **Show AD Info** to see the availability domain information.
- You can join a maximum of seven peers from each domain.

See Create a Channel.

You must be an administrator to perform this task.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the peer node that you want to add to a channel.
3. In the Node Information page, click the Channels pane to view the list of channels the peer is already using.
4. Click **Join New Channels**.

   The Join New Channels dialog is displayed.
5. Click the **Channel Name** field and from the list select the name of the channel to join. Click the field again to select another channel. Click **Join**.

Add an Anchor Peer

Each member using a channel must designate at least one anchor peer. Anchor peers are primary network contact points, and are used to discover and communicate with other network peers on the channel.

You can designate one or more peers in your organization as an anchor peer on a channel. For a high availability network, you can specify two or more anchor peers. All members using the network channel must use their console to designate one or more of their peer nodes as anchor peers.

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.

   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the channel name you want to add anchor peers to.

   The Channel Information page is displayed.
3. In the Channel Information page, click the Peers pane.
4. Locate the peer or peers that you want to designate as anchor peers and click their **Anchor Peer** checkboxes to select them.
5. Click the **Apply** button.

Change or Remove an Anchor Peer

(New in 19.2.1) You can change or remove a channel's anchor peers. Anchor peers are primary network contact points, and are used to discover and communicate with other network peers on the channel.

Before you change or remove the channel's anchor peers, note the following information:

- To communicate on the channel, you must designate one or more peers in your organization as an anchor peer.
- For a high availability network, you can specify two or more anchor peers.
- All members using the network channel must use their console to designate one or more of their peer nodes as anchor peers.

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.

   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the channel name you want to remove anchor peers from.

   The Channel Information page is displayed.
3. In the Channel Information page, click the Peers pane.
4. Locate the peer or peers that you want to remove as anchor peers and clear their **Anchor Peer** checkboxes. Alternatively, to add another peer as an anchor peer, click its **Anchor Peer** checkbox to select it.
5. Click the **Apply** button.

View Information About Instantiated Chaincodes

You can view information about the chaincodes instantiated on the different channels in your network.

Some examples of when you need information about instantiated chaincodes are to determine if you need to upgrade the chaincode, or to find out which channels the chaincode was instantiated on.

1. Go to the console and select the Channels tab.
2. In the channels table, click the channel name with the chaincode that you want to view information for.
3. In the Channel Information page, confirm that the Instantiated Chaincodes pane is selected
4. In the chaincode table, you can:
   - Click the chaincode to go to the Chaincodes tab to learn more information about it, for example the peers that the chaincode is installed on and the channels that the chaincode is instantiate on.
   - In a chaincode's More Actions menu, click **View Endorsement Policy** to find details about the chaincode's endorsement policy, for example who must endorse the chaincode and the signed by expression string.
5. (Optional) If you see a channel listing without a chaincode, then you can go to the Chaincodes tab and instantiate a chaincode to the channel. See Instantiate a Chaincode.

Work With Channel Policies and ACLs

(19.1.3 and later versions only) This topic contains information about a channel's policies and ACLs.

Topics:

- What Are Channel Policies?
- Add or Modify a Channel's Policies
- Delete a Channel's Policies
- What Are Channel ACLs?
- Update Channel ACLs

What Are Channel Policies?

A policy defines a set of conditions. The required parties must meet the policy's conditions before their signatures are considered valid and the corresponding request happens on the network.

The blockchain network is managed by these policies. Policies check the identity associated with a request against the policy associated with the resource needed to fulfill the request. Policies are located in the channel's configuration.

After you configure the channel's policies, you assign them to the channel's ACLs resources to determine which members are required to sign before a change or action can happen on the channel. For example, suppose you modified the Writers policy to include members from Organization A or Organization B. Then you assigned the Writers policy to the channel's cscc/GetConfigBlock ACL resource. Now only a member from Organization A or Organization B can call GetConfigBlock on the cscc component.

What Are the Policy Types?

There are two policy types: Signature and ImplicitMeta.

- **Signature** — Specifies a combination of evaluation rules. It supports combinations of *AND*, *OR*, and *NOutOf*. For example, you could define something like “An admin of org A and 2 other admins" or "11 of 20 org admins.”
  Note that when you modify the Oracle Blockchain Platform's default Admins policy, which was created as an ImplicitMeta policy, you'll use the Signature policy. Any new policies you create will be Signature policies.
- **ImplicitMeta** — This policy type is only valid in the context of configuration. It aggregates the result of evaluating policies deeper in the configuration hierarchy, which are defined by Signature policies. It supports default rules, for example “A majority of the organization admin policies.”
  Oracle Blockchain Platform uses the ImplicitMeta policy type to create the Admins policy. When you modify the Admins policy, you'll use the Signature policy. You can't create or modify any policies using the ImplicitMeta policy. Oracle Blockchain Platform only supports modifying or creating policies using the Signature policy type.

When Are Policies Created?

When you add a channel to the network, Oracle Blockchain Platform created new default policies. The default policies are: Admins (ImplicitMeta policy), Creator, Writers, and Readers (Signature policies). If needed, you can modify these policies or create new policies.

Note the following important issues about channel policies:

- You can use the console to create a channel and set your organization's ACL to ReaderOnly. After you save the new channel, you can't update this ACL setting from the channel's Edit Organization option.

  However, you can use the console's Manage Channel Policies functionality to add your organization to the Writers policy, which overwrites the channel's ReaderOnly ACL setting.
- When you use the Hyperledger Fabric SDKs to create a channel, Fabric uses the ImplicitMeta policies as the default channel policies for Readers and Writers. When the channel uses these policies, the Oracle Blockchain Platform console can't guarantee that the administrative operations (for example, edit organization) will be successfully processed.

  To correct this issue, update the readers and writers policies to Signature policies, and define the policy rules as needed. See https://hyperledger-fabric.readthedocs.io/en/release-1.3/access_control.html
- When you use the Hyperledger Fabric SDKs or CLI to create a channel, the Creator policy isn't included in the configtx.yaml file. The Creator policy is required by Oracle Blockchain Platform to allow the channel creator to edit a channel's configuration. You must manually edit the configtx.yaml file and add the Creator policy.

Add or Modify a Channel's Policies

(19.1.3 and later versions only) You can add or modify a channel's policy to specify which members are required to perform a specific action on the channel. After you define policies, you assign them to the channel's ACLs.

Before you add or update policies, you need to understand how Oracle Blockchain Platform creates default channel policies. See What Are Channel Policies?

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.
   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the channel name that you want to add policies to or modify policies for.
   The Channel Information page is displayed.
3. In the Channel Information page, click the Channel Policies pane.
4. Do one of the following:
   - To add a new policy, click the **Create a New Policy** button. The **Create Policy** dialog is displayed. Enter a name in the **Policy Name** field and select Signature in the **Policy Type** field. Expand the **Signature Policy** section.
   - To modify an existing policy, click a policy's name. The **Update Policy** dialog is displayed.
5. Click the **Add Identity** button to add an organization. Or modify an existing signature policy as needed. Note the following information:

Field	Description
MSP ID	From the dropdown menu, select the organization that must sign the policy.
Role	Select the corresponding peer role required by the policy. Usually this will be member. You can find a peer’s role by viewing its configuration information.
Policy Expression Mode	In most cases, you’ll use **Basic**. Select **Advanced** to write an expression string using *AND*, *OR*, and *NOutOf*. See the Hyperledger Fabric documentation for information about how to write a valid policy expression string.
Signed By	Select how many members must sign the policy to fulfill the request.

6. If you're adding a new policy, then click **Create**. If you're modifying a policy, then click **Update**.

Delete a Channel's Policies

(19.1.3 and later versions only) You can delete a policy from a channel.

You can't delete a channel policy if it is assigned to an ACL. Before you try to delete a channel policy, confirm that the policy isn't assigned.

You must be an administrator to perform this task.

1. Go to the console and select the Channels tab.
   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the channel that you want to delete a policy from.
   The Channel Information page is displayed.
3. In the Channel Information page, click the Channel Policies pane.
4. Locate the policy that you want to delete and click its **More Options** button.
5. Click **Remove** and confirm the deletion.

What Are Channel ACLs?

(19.1.3 and later versions only) Access control lists (ACLs) use policies to manage which organizations and roles can access a channel's resources.

Users interact with the blockchain network by targeting components such as the query system chaincode (qscc), lifecycle system chaincode (lscc), configuration system chaincode (cscc), peer, and event. These components are associated with specific resources (for example, GetConfigBlock or GetChaincodeData) that you can assign policies to at the channel level. These policies are a part of the channel's configuration.

A policy defines which organizations and roles can request a resource. When a request is made, the policy tells the system to check the requester's identity and determine if it's authorized to make the request. When you create a channel, Oracle Blockchain Platform includes the default Hyperledger Fabric ACLs with the channel. Oracle Blockchain Platform also creates four default policies (Admin, Creator, Writers, and Readers) for the channel. You can modify these policies or create new policies as needed. See What Are Channel Policies?

Update Channel ACLs

(19.1.3 and later versions only) You can update the channel's ACLs by assigning policies to the channel's resources. A policy defines which organizations and roles can request a resource

Before you update a channel's ACLs, you should understand what policies and ACLs are. See What Are Channel Policies? and What Are Channel ACLs?

1. Go to the console and select the Channels tab.
   The Channels tab is displayed and the channel table contains a list of all of the channels on your network.
2. In the channels table, click the name of the channel that you want to update ACLs for.
   The Channel Information page is displayed.
3. In the Channel Information page, click the ACLs pane.
4. In the Resources table, locate the resource that you want to update. Click the resource's **Expand** button and select the policy that you want to assign to the resource.
5. Modify the other resource's policies as needed.
6. Click **Update ACLs**.

Add or Remove Orderers To or From a Channel

The orderer admin organization can add or remove orderers from a channel.

To add orderers to a channel:

1. In the founder console, open the Channels tab and select the channel to see its details view.
2. Open the Orderers subtab. All orderer nodes currently joined to the channel are listed.
3. Click **Join Channel**. Select an OSN not yet in this channel and click **Join**.

To remove orderers from a channel:

1. In the founder console, open the Channels tab and select the channel to see its details view.
2. Open the Orderers subtab. All orderer nodes currently joined to the channel are listed.
3. Select the orderer you want to remove from the channel and from its More Actions menu select **Remove**.

Set the Orderer Administrator Organization

You can assign the administration of OSNs in a channel to any organization. Normally either the founder or the channel creator would be assigned.

1. In the founder console, open the Channels tab.
2. Select the channel for which you want to set the orderer administrator organization, and from the Action menu select **Manage OSNs Admin**.
3. Select from the list of available organizations, and click **Submit**.

Edit Ordering Service Settings for a Channel

You can update the ordering service settings for a particular channel.

Note the following important information about editing the ordering service settings for a channel:

- Separately you can update the ordering service settings for the entire network as described in Edit Ordering Service Settings for the Network.
- If you change the ordering service settings and there are applications running against the network, then those applications must be manually updated to use the revised ordering service settings.
- It isn’t common, but in some situations, you might expose a different ordering service to some of the network participants. In this case, you’ll export the updated network config block and the required participants will import the revised settings. See Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service.

You must be an administrator to perform this task.

1. Go to the founder’s console and select the Channels tab.

2. Locate the channel, click the **More Actions** menu, and select **Update Ordering Service Settings**.

   The Ordering Service Settings dialog is displayed.

3. Update the settings as needed.

Field	Description
Batch Timout (ms)	Specify the amount of time in milliseconds that the system should wait before creating a batch. Enter a number between 1 and 3600000.
Max Message Count	Specify the maximum number of message to include in a batch. Enter a number between 1 and 4294967295.
Absolute Message Bytes	Specify the maximum number of bytes allowed for the serialized messages in a batch. This number must be larger than the value you enter in the Preferred Message Bytes field.
Preferred Message Bytes	Specify the preferred number of bytes allowed for the serialized messages in a batch. A message larger than this size results in a larger batch, but the batch size will be equal to or less than the number of bytes you specified in the Absolute Message Bytes field. Oracle recommends that you set this value to 1 MB or less. The value that you enter in this field must be smaller than the value you enter in the Absolute Message Bytes field.
Snapshot Interval Size	Defines number of MB per which a snapshot is taken.

4. Click **Update**.

   The updated settings are saved.

Manage Certificates

This topic contains information about how to manage your network's certificates.

Topics:

- Typical Workflows to Manage Certificates
- Export Certificates
- Import Certificates to Add Organizations to the Network
- What's a Certificate Revocation List?
- View and Manage Certificates
- Revoke Certificates
- Apply the CRL

Typical Workflows to Manage Certificates

Here are the common tasks for managing your network's certificates.

Adding Organizations to the Network

You must be an administrator to perform these tasks.

Task	Description	More Information
Export or prepare an organization's certificates	The organization that wants to join the network either outputs or writes its certificates file and gives it to the founder.	Export Certificates Create a Fabric Organization's Certificates File Create an Organization's Third-Party Certificates File
Import member certificates	The founder imports the organization's certificates file to add the organization to the network.	Import Certificates to Add Organizations to the Network
View certificates	The founder can view and manage the network's certificates.	View and Manage Certificates

Revoking Certificates

You must be an administrator to perform these tasks.

Task	Description	More Information
Decide which certificates to revoke	View the certificates on your system to determine which ones to revoke to keep the network secure.	View and Manage Certificates
Select the certificates to revoke	Revoke the certificates in your CA.	Revoke Certificates
Apply CRL	Generates and applies an updated CRL to ensure that clients with revoked certificates can't access channels.	Apply the CRL

Export Certificates

Founders and participant organizations must import and export certificate JSON files to create the network.

Note the following information:

- For the founder to add a participant organization to the blockchain network, the participant must export its certificates file and make it available to the founder. The founder then uploads the certificates file to add the participant organization to the network.
- The certificate export file contains admincerts, cacerts, and tlscacerts.

- You might need to export certificates for blockchain or application developers. For example, a client application needs the TLS certificate to interact with peers or orderers.

For information about writing certificate files required to add Hyperledger Fabric or Third-Party organizations to the network, see Extend the Network.

1. Go to the console and select the Network tab.
2. In the Network tab, go to the Organizations table, locate the member that you want to export certificates for, and click its **More Actions** button.
3. Click **Export Certificates**.

   Note that files exported by the console and REST APIs are only compatible for import with the same component. That is you can't successfully use the REST API to import an export file created with the console. Likewise, you can't successfully use the console to import an export file created with the REST API.
4. Specify where to save the file. Click **OK** to save the certificates file.
5. Send the certificates JSON file to the founder for import. See Import Certificates to Add Organizations to the Network.

Import Certificates to Add Organizations to the Network

To add an organization to the network, the founder must import a certificates file that was exported or prepared by the organization that wants to join the network.

You can import certificates for the following organization types.

Type	Description
Oracle Blockchain Platform Participant Organization	You can import a participant organization into a Oracle Blockchain Platform network. You upload the certificates that the participant organization exported from the console and sent to you. For information about creating certificates for upload and a list of the other steps that you need to perform to successfully set up a participant organization on the network, see Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service.
Hyperledger Fabric Organization	You can import a Hyperledger Fabric organization into an Oracle Blockchain Platform network. To successfully upload a Fabric organization’s certificates file, you must modify the certificates file to replace all instances of \n with the newline character. See Typical Workflow to Join a Fabric Organization to an Oracle Blockchain Platform Network.
Third-Party Certificate Organization	You can import an organization that is using certificates generated from a third-party CA server. To successfully upload a third-party organization’s certificates file, you must modify the certificates file to replace all instances of \n with the newline character. See Typical Workflow to Join an Organization with Third-Party Certificates to an Oracle Blockchain Platform Network.

You must be an administrator to import certificates.

1. Go to the console and select the Network tab.
2. In the Network tab, click **Add Organizations**. The Add Organizations page is displayed.

   Note that files exported by the console and REST APIs are only compatible for import with the same component. That is you can't successfully use the REST API to import an export file created with the console. Likewise, you can't successfully use the console to import an export file created with the REST API.
3. Click **Upload Organization Certificates**. The File Upload dialog is displayed.
4. Browse for and select the JSON file containing the certificate information for the organization you want to add to the network. Usually this file is named `certs.json`. Click **Open**.
5. (Optional) Click the plus (+) icon to locate and upload another organization’s certificate information.
6. Click **Add**. The organizations that you added are displayed in the Organization table.

   Note the following information for Oracle Blockchain Platform participant, Hyperledger Fabric, and third-party certificate organizations. Even though the founder uploaded the certificate information, the added organization can’t use the ordering service to communicate on the network until it imports the founder’s ordering service settings. The founder must export its ordering service settings and give the resulting file to the joining organizations for import. See one of the following:

   - For Oracle Blockchain Platform participants, see Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service.
   - For Hyperledger Fabric organizations, see Prepare the Fabric Environment to Use the Oracle Blockchain Platform Network.
   - For third-party certificate organizations, see Prepare the Third-Party Environment to Use the Oracle Blockchain Platform Network.

What's a Certificate Revocation List?

You use a certificate revocation list (CRL) to help manage the certificates throughout your network.

A CRL is a list of digital certificates that the issuing Certificate Authority (CA) has revoked before their scheduled expiration date and should no longer be trusted and used on the network. For example, you should revoke any certificates that have been lost, stolen, or compromised.

After you use the Manage Certificates functionality to revoke certificates for users, Oracle Blockchain Platform creates the CRL. To ensure that the certificates are revoked throughout the network, you’ll need to:

- Use the Apply CRL functionality after you join peers to a channel created by another network member. Apply CRL prevents clients with revoked certificates from accessing the channel. See Apply the CRL.

View and Manage Certificates

Use the console to view and manage the user certificates in your instance and any of the certificates you imported when building the network.

1. Go to the console and select the Network tab.
2. In the Network tab, locate your organization’s ID and click its **More Actions** button. Select **Manage Client Certificates**.

   Note that the Certificate Summary table will be empty until you add users to your instance. Also, the administrator’s certificate doesn’t display in this table. This is to prevent you from accidentally revoking the administrator’s certificate.

   Organizations with third-party certificates or Hyperledger Fabric organization with revoked certificates won't display in this table. In such cases, you must use the native Hyperledger Fabric CLI or SDK to import the organization's certificate revocation list (CRL) file.

   The Certificates Summary dialog is displayed and shows a list of the certificates in your instance.
3. As needed, perform any of the following tasks:
   - Revoke certificates. See Revoke Certificates.
   - If you’ve revoked certificates and are working in a network with multiple members, then use Apply CRL after you join peers to a channel created by another network member. Apply CRL prevents clients with revoked certificates from accessing the channel. See Apply the CRL.

Revoke Certificates

An organization can revoke certificates for any of its users. To ensure that the network remains secure, you should revoke certificates in case they’re lost, stolen, or compromised.

You must be an administrator to perform this task.

1. Go to the console and select the Network tab.
2. In the Network tab, locate your organization’s ID and click its **More Actions** button. Select **Manage Client Certificates**.

   The Certificates Summary dialog is displayed.
3. In the Certificates Summary dialog, locate and select the IDs of the users that you want to revoke certificates for.
4. Click Revoke and confirm that you want to permanently revoke certificates for the selected users.

   The users with revoked certificate display in the table and are added to the CRL.
5. If you’re working in a network with other members, then to ensure that their revoked certificates are cleaned up across the network, you must do the following:
   - If you’re working in a network with multiple members, then apply the CRL after you join peers to a channel created by another network member. Apply CRL prevents clients with revoked certificates from accessing the channel. See Apply the CRL.

Apply the CRL

If you're working in a network, then you must apply the CRL after you join peers to a channel created by another network member. Apply CRL prevents members with revoked certificates from accessing the channel.

You must do the following tasks before applying the CRL:

- Revoke certificates. See Revoke Certificates

You must be an administrator to perform this task.

1. Go to the console and select the Network tab.
2. In the Network tab, locate your organization’s ID and click its **More Actions** button. Select **Manage Client Certificates**.

   The Certificates Summary dialog is displayed.
3. Click the Apply CRL button and confirm that you want to apply the CRL.

Manage Ordering Service

This topic contains information about how founders and participants manage the ordering service.

Topics:

- What is the Ordering Service?
- Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service
- Edit Ordering Service Settings for the Network
- View Ordering Service Settings

Additionally, several channel-specific tasks for the orderer nodes can be performed on the Channels page of the console. See:

- Add or Remove Orderers To or From a Channel
- Set the Orderer Administrator Organization
- Edit Ordering Service Settings for a Channel

What is the Ordering Service?

In Oracle Blockchain Platform v20.3.1 and later, we support Raft as our consensus type.

For more information on the Raft protocol, see: The Ordering Service - Raft.

With the older Kafka consensus type, the whole network can have at most two orderer nodes, and they have to join all channels. In some cases, they may be overloaded, and cannot be scaled out. With the Raft consensus type, the network can have an arbitrary number of orderer nodes, and each channel can define its own orderer node set. Different channels can use different orderer nodes, and orderer nodes will no longer be the bottleneck.

However, the Raft consensus type can be complicated to configure properly. There are rules about what can or can't be done, and if these rules are not followed the channel and even the network may not work. The following guidelines should reduce the problems you encounter:

Keep the Majority of the Ordering Service Nodes (OSN) Alive

The Raft consensus algorithm requires that the majority of ordering service nodes (OSNs) are working; otherwise no consensus can be made. Majority means greater than 50%. For example, for five OSNs, there must be at least three OSNs working; for six OSNs, there must be at least four OSNs working.

- If there are 50% or less OSNs working in the network, network management will no longer be functional. No new channels can be created, no new orderer nodes can be added into network, no orderer can be removed from network, and so on.
- If there are 50% or less OSNs working in the application channel, no transaction can be submitted to this application channel. Queries may still function correctly, however administrative operations such as adding a new organization, changing the access control list, or instantiating chaincodes will fail.

Be cautious when adding a new OSN to the network or an application channel. Ensure the owner is trustworthy and the OSN is robust.

When removing OSNs or an organization, ensure that more than 50% of the OSNs will remain working. For example, if you had 2 organizations with 3 OSNs each, if you removed one organization, during the removal it would be interpreted as only 50% of the OSNs being functional. Add an OSN to the remaining organization before deleting the extraneous organization to ensure that you always exceed 50% of the OSNs working.

Do Not Add or Remove Orderers Frequently

Every time a new OSN is added into a network or channel, or an existing OSN is removed from a channel, the current Raft OSN cluster will briefly become unstable. During this period, no transactions can be handled, and an error message similar to the following may indicate such a status:

```
UNKNOWN: Stream removed
SERVICE UNAVAILABLE
BAD REQUEST
```

This may last a few minutes. If you have removed the previous Raft leader OSN from the channel, this may last as long as 20 minutes.

Ensure that you aren't adding or removing orderers frequently. If multiple orderers must be added or removed, do one at a time ensuring that the network has returned to operational status before making the next change.

Ensure the New Orderer is Started As Soon As Possible

When adding a new orderer into network, usually two organizations will be involved: the founder and the owner of the new orderer. Both parties must follow the instructions in Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service all the way to completion or the founder won't be able to manage the network.

Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service

When you provision a participant instance, it is created with 3 orderers. There orderers are inactive until they are joined to a network. When you scale out a founder, the new orderers are also inactive until they are joined to a network.

If multiple orderers must be added or removed, do one at a time ensuring that the network has returned to operational status before making the next change. See What is the Ordering Service? for additional important details about adding, removing, starting, and stopping Raft orderers.

Export the OSN Settings From the Participant or Scaled-Out Orderers

To join the participant or scaled-out orderers to a network, you need to export their settings and import them into the founder.

1. In the participant console (or the founder console for scaled-out orderers), on the Node tab find the orderer node (or the first orderer node if multiple nodes exist). Select the Action menu for this node and select **Export OSN Settings**.

   This will generate a JSON file with the settings and save the file. The file contains the organization's certificate and the selected orderer service node (OSN) settings signed by the private key of the administrator of the participant organization. This file needs to be sent to the administrator of the founder instance.

   Applications being run on channels using this OSN also require this exported TLS certificate. See Before You Develop an Application.

2. In the founder console, open the Network tab. Click **Add OSN**. A window opens prompting you for the location of the JSON file provided by the participant. Select to upload the file and click **Add**.

   The participant organization or newly scaled-out orderer will be added to the orderer organization section of the system channel list.

Export the Founder's Configuration Settings

Once the participant or scaled-out orderers have been added to the founder, you need to export the founder's settings and import them to the participant or scaled-out orderer.

1. In the founder console, open the Network tab. Click **Export Network Config Block**.

   The network configuration block contains the latest system channel configuration block. This can be saved and sent to the participant administrator.

2. In the participant console (or the founder console for scaled-out orderers), on the Node tab find the orderer node (or the first orderer node if multiple nodes exist). Select the Action menu for this node and select **Import Network Config Block**.

   You'll be prompted for the file sent by the founder instance administrator.

3. In the participant console, refresh the Node tab. The orderer node status should be listed as "down". From the Action menu select **Start**.

   Each orderer node started will be added to the Raft cluster in the founder.

Each time a new OSN is added by scaling out the orderer (as described in Scale Your Instance) these steps need to be repeated to add the new OSN to the Raft cluster.

> **Note:**
>
> You can't add multiple OSNs into a network in a single batch. Ensure only 1 OSN is added at a time.

Edit Ordering Service Settings for the Network

You can update the ordering service settings for the founder instance.

Note the following important information about editing the ordering service settings:

- The updated settings are used when you create new channels and are not applied to existing channels.
- Separately you can update the ordering service settings for an individual existing channels as described in Edit Ordering Service Settings for a Channel.
- If you change the ordering service settings and there are applications running against the network, then those applications must be manually updated to use the revised ordering service settings.
- It isn't common, but in some situations, you might expose a different ordering service to some of the network participants. In this case, you'll export the updated network config block and the required participants will import the revised settings. See Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service.

You must be an administrator to perform this task.

1. Go to the founder's console and select the Network tab.
2. Click the **Ordering Service Settings** button.

   The Ordering Service Settings dialog is displayed.
3. Update the settings as needed.

Field	Description
Batch Timout (ms)	Specify the amount of time in milliseconds that the system should wait before creating a batch. Enter a number between 1 and 3600000.
Max Message Count	Specify the maximum number of message to include in a batch. Enter a number between 1 and 4294967295.
Absolute Message Bytes	Specify the maximum number of bytes allowed for the serialized messages in a batch. This number must be larger than the value you enter in the Preferred Message Bytes field.

Field	Description
Preferred Message Bytes	Specify the preferred number of bytes allowed for the serialized messages in a batch. A message larger than this size results in a larger batch, but the batch size will be equal to or less than the number of bytes you specified in the Absolute Message Bytes field. Oracle recommends that you set this value to 1 MB or less. The value that you enter in this field must be smaller than the value you enter in the Absolute Message Bytes field.
Snapshot Interval Size	Defines number of MB per which a snapshot is taken.

4. Click **Update**.

   The updated settings are saved.

View Ordering Service Settings

You can view the founder's ordering service settings that were imported into a participant’s Oracle Blockchain Platform instance.

If the founder changes the ordering service settings the new settings must be ported to the participant as described in Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service. If there are applications running against the network, then those applications must be manually updated to use the revised ordering service settings.

1. Go to the participant’s console and select the Network tab.
2. Click **Ordering Service Settings** and click **View**.

   The Ordering Settings dialog is displayed.

4

Understand and Manage Nodes by Type

This topic contains information to help you understand the different node types and where you can get more information about how the nodes are performing in the network.

Topics:

- Manage CA Nodes
- Manage the Console Node
- Manage Orderer Nodes
- Manage Peer Nodes

Manage CA Nodes

This topic contains information about certificate authority (CA) nodes.

Topics

- View and Edit the CA Node Configuration
- View Health Information for a CA Node

View and Edit the CA Node Configuration

A certificate authority (CA) node's configuration determines how the node performs and behaves on the network.

Only administrators can change a node's configuration. If you've got user permissions, then you can view a node's configuration settings. See CA Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the CA node that you want configuration information for, and click the node's **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node's settings as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.
6. Restart the node to apply any changes that you made.

View Health Information for a CA Node

You can check a certificate authority (CA) node’s metrics to see how the node is performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health pane displays the node’s performance metrics: CPU utilization and memory utilization.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the name of the CA node you want to see health information for.

   The Node Information page is displayed.
3. Click the **Health** pane to view the node’s performance metrics.

   If the utilization percentages are consistently high, then contact Oracle Support.

Manage the Console Node

This topic contains information about the console node.

Topics:

- View and Edit the Console Node Configuration
- View Health Information for the Console Node

View and Edit the Console Node Configuration

The console node’s configuration determines how it performs and behaves on the network.

Only administrators can change a node’s configuration. If you've got user permissions, then you can view a node’s configuration settings. See Console Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the console node and click its **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node’s settings as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.
6. Restart the node to apply any changes that you made.

View Health Information for the Console Node

You can check the console node’s metrics to see how it's performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health Overview pane displays these performance metrics: CPU utilization and memory utilization.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the console node.

   The Node Information page is displayed.
3. Click the **Health Overview** pane to view the node’s performance metrics.

   Note the following information:

   - If the CPU Utilization percentage is too high, then it might be because too many users are trying to access the console at the same time, or that the console is having technical issues.
   - If the utilization percentages are consistently high, then contact Oracle Support

Manage Orderer Nodes

This topic contains information about orderer nodes.

Topics

- View and Edit the Orderer Node Configuration
- View Health Information for an Orderer Node
- Add an Orderer Node

View and Edit the Orderer Node Configuration

An orderer node’s configuration determines how the node performs and behaves on the network.

Only administrators can change a node’s configuration. If you've got user permissions, then you can view a node’s configuration settings. See Orderer Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the orderer node that you want configuration information for, and click the node’s **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node’s settings as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.

6. Restart the node to apply any changes that you made.

View Health Information for an Orderer Node

You can check an orderer node’s metrics to see how the node is performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health Overview pane displays these performance metrics: CPU utilization and memory utilization.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the name of the orderer node you want to see health information for.

   The Node Information page is displayed.
3. Click the **Health** pane to view the node’s performance metrics.

   If the utilization percentages are consistently high, then contact Oracle Support. If the Disk Utilization percentage is too high, then the ledger might not get stored on the node properly.

Add an Orderer Node

Founder instances are provisioned with 3 OSNs, all of which are active after instance creation. Additional OSNs can be scaled out as described in Scale Your Instance. These OSNs will not be started automatically. You must start them and export the updated network configuration block to the partipant instances as described in Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service.

Participant instances are created with 3 OSNs, but none of these OSNs are joined to the network or started when the instance is provisioned. You must follow the instructions in Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service in order to join them to the network and start the nodes. If you want to scale out the participant OSNs these steps must be repeated.

Manage Peer Nodes

This topic contains information about peer nodes.

Topics

- View and Edit the Peer Node Configuration
- List Chaincodes Installed on a Peer Node
- View Health Information for a Peer Node
- Export and Import Peer Nodes

View and Edit the Peer Node Configuration

A peer node’s configuration determines how the node performs and behaves on the network.

Only administrators can change a node’s configuration. If you've got user permissions, then you can view a node’s configuration settings. See Peer Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the peer node that you want configuration information for, and click the node’s **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

   The Configure dialog is displayed.
4. If you're an administrator, then modify the node’s settings as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.
6. Restart the node to apply any changes that you made.

List Chaincodes Installed on a Peer Node

You can view a list of the chaincodes and their versions installed on a specific peer node in your network.

If you don’t see the chaincode or the chaincode version you were expecting, then you can install a chaincode or upgrade a chaincode to the peer node. You must be an administrator to install or upgrade a chaincode.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the name of the peer node you want to see information for.

   The Node Information page is displayed.
3. Click the **Chaincodes** pane to view a list of chaincodes installed on the selected peer node.

View Health Information for a Peer Node

You can check a peer node’s metrics to see how the node is performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health Overview pane displays these performance metrics: CPU utilization, memory utilization, user transactions endorsed, and user transactions committed.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the name of the peer node you want to see health information for.

   The Node Information page is displayed.
3. Click the **Health Overview** pane to view the node’s performance metrics.

Note the following information:

- If the CPU Utilization and Memory Utilization percentages are too high, then it might be because the peer is overloaded with endorsement requests. Consider adding another peer or changing the endorsement policy.
- If the Disk Utilization percentage is too high, then the ledger might not get stored on the node properly.
- The User Transactions Endorsed and User Transaction Committed metrics are collected and refreshed every ten minutes. The counts you see are cumulative.
- If the utilization percentages are consistently high, then contact Oracle Support.

Export and Import Peer Nodes

If you want to run the blockchain transactions through the REST proxy, then after you've added a participant to the network, you must export its peer nodes and import them into the founder.

You need to do this export and import step because the REST proxy's end point configuration needs to know about the peers from both members. After you've completed this step then you'll have to update the founder and participants' REST proxy nodes to add the peers so that the requests can be routed as required by the endorsement policy.

See Typical Workflow to Join a Participant Organization to an Oracle Blockchain Platform Network.

1. Go to the participant's console and select the Nodes tab.
2. Click **Export/Import Peers** and select **Export**.

   The Export Nodes dialog is displayed.
3. In the Peer List field, select the peer nodes that you want to export. Click **Export**.

   Note that files exported by the console and REST APIs are only compatible for import with the same component. That is you can't successfully use the REST API to import an export file created with the console. Likewise, you can't successfully use the console to import an export file created with the REST API.
4. To import, go to the founder's Oracle Blockchain Platform console and select the Nodes tab.
5. Click **Export/Import Peers** and select **Import**.

   The Import Remote Nodes dialog is displayed.
6. Click **Upload remote nodes configurations** and browse for and select the JSON file containing the node configuration information. Usually this file is named `<instance name>-exported-nodes.json`.

   Note that files exported by the console and REST APIs are only compatible for import with the same component. That is you can't successfully use the REST API to import an export file created with the console. Likewise, you can't successfully use the console to import an export file created with the REST API.
7. Click the plus icon to upload another node configuration file for import.
8. Click **Import**.

9. To confirm that the nodes were added successfully, you can:
   - Go to the founder’s Nodes tab and in the nodes table locate the names of the imported peer nodes. Note that the imported nodes type is Remote Peer. You can’t view or edit a remote peer’s configuration information.
   - Go to the founder’s Network tab and click **Topology View** and locate the names of the imported peer nodes.

Manage REST Proxy Nodes

This topic contains information to help you understand, set up, and manage the REST proxy nodes.

Topics

- How's the REST Proxy Used?
- Add Enrollments to the REST Proxy
- View and Edit the REST Proxy Node Configuration
- View Health Information for a REST Proxy Node

How's the REST Proxy Used?

The REST proxy maps an application identity to a blockchain member, which allows users and applications to call the Oracle Blockchain Platform REST APIs.

Instead of using the native Hyperledger Fabric APIs, Oracle Blockchain Platform can use the REST proxy to interact with the Hyperledger Fabric network. When you use the native Hyperledger Fabric APIs, you connect to the peers and orderer directly. However, the REST proxy allows you to query or invoke a Fabric chaincode through the RESTful protocol.

Add Enrollments to the REST Proxy

Enrollments allow users to call the REST proxy without an enrollment certificate. Enrollements require a new user group to be defined on your authentication server.

Adding Enrollments When Using Microsoft Active Directory as Your Authentication Server

Adding an enrollment to the REST Proxy requires a new user group to be added to Active Directory: `<Rest Proxy Client Users group name>_<custom enrolment name>`. You can then use the Blockchain Platform console to map the enrollment to this group.

1. Create a new Active Directory group called `<Rest Proxy Client Users group name>_<custom enrolment name>`.
2. Add any users needing to use the custom enrollment to this group.
3. Go to the Blockchain Platform console and select the Nodes tab.
4. In the Nodes tab, find the REST proxy node you want to add an enrollment to and open the **More Actions** menu.
5. Click **View or create enrollments** to see a list of the node’s current enrollments.

6. Click **Create New Enrollment**.
7. In the **User Name** field, enter `<custom enrolment name>` from the first step. Note that this is case-sensitive and must match the user group you created. Click **Enroll**.
    - The enrollment is created and displays in the Enrollments table.
    - The new enrollment is copied to each REST proxy node in the network.

Adding Enrollments When Using OpenLDAP or Oracle Internet Directory as Your Authentication Server

Adding an enrollment to the REST Proxy creates a new user role in the `OBP_<platform-name>_<instance-name>_REST_<custom-enrollment>` group on your LDAP server.
After the enrollment is created in the console, the Administrator uses the LDAP server to assign the required users to this role.

For information about how users access the REST resources, see REST API for Oracle Blockchain Platform.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, find the REST proxy node you want to add an enrollment to and open the **More Actions** menu.
3. Click **View or create enrollments** to see a list of the node's current enrollments.
4. Click **Create New Enrollment**.

    The Create New Enrollment dialog is displayed.
5. In the **User Name** field, enter a name for the enrollment. Click **Enroll**.

    After you click **Enroll**:
    - The enrollment is created and displays in the Enrollments table.
    - The new enrollment is copied to each REST proxy node in the network.
    - A new user role in the `OBP_<platform-name>_<instance-name>_REST_<custom-enrollment>` group on your LDAP server.

View and Edit the REST Proxy Node Configuration

A REST proxy node's configuration determines how the node performs and behaves on the network.

Only administrators can change a node's configuration. If you've got user permissions, then you can view a node's configuration settings. See REST Proxy Node Attributes.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, go to the Nodes table, locate the REST proxy node that you want configuration information for, and click the node's **More Actions** button.
3. The configuration option is determined by your permissions. If you're an administrator, locate and click **Edit Configuration**. If you're a user, locate and click **View**.

    The Configure dialog is displayed.

4. If you're an administrator, then modify the node’s **Proposal Wait Time (ms)** and **Transaction Wait Time (ms)** attributes as needed.
5. Click **Submit** to save the configuration changes, or click **X** to close the Configure dialog.

View Health Information for a REST Proxy Node

You can check a REST proxy node’s metrics to see how the node is performing on the blockchain network. This information helps you discover and diagnose performance problems.

The Health pane displays these performance metrics: CPU utilization and memory utilization.

1. Go to the console and select the Nodes tab.
2. In the Nodes tab, click the REST proxy node you want to see health information for.

   The Node Information page is displayed.
3. Click the **Health** pane to view the node’s performance metrics.

   If the utilization percentages are consistently high, then contact Oracle Support.

5

Extend the Network

This topic contains information to help founders add organizations to the blockchain network. This topic also contains information to help organizations join a network.

Topics

- Add Oracle Blockchain Platform Participant Organizations to the Network
- Add Fabric Organizations to the Network
- Add Organizations with Third-Party Certificates to the Network

Add Oracle Blockchain Platform Participant Organizations to the Network

This topic contains information about joining an Oracle Blockchain Platform participant organization to an Oracle Blockchain Platform network.

Topics:

- Typical Workflow to Join a Participant Organization to an Oracle Blockchain Platform Network

Typical Workflow to Join a Participant Organization to an Oracle Blockchain Platform Network

Here are the tasks the founder and participants organizations need to perform to set up a blockchain network.

Adding Participant Organizations to a Blockchain Network

Task	Who Does This?	Description	More Information
Export the participant organization's certificates and import them into the network	Participant organization outputs certificates Founder organization uploads certificates	In the participant organization's instance, use the wizard to output the certificates into a JSON file and send them to the founder organization. The founder uploads the certificates to add the participant to the network.	Import Certificates to Add Organizations to the Network

Task	Who Does This?	Description	More Information
Export the participant organization's ordering service node (OSN) settings and send to the founder administrator	Participant organization outputs a settings file Founder organization uploads the settings	In the participant organization's instance, export the settings into a JSON file and sends them to the founder organization. The founder uploads the settings to add the ordering service.	Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service
Export the founder organization's network configuration block and upload it to the participant organization	Founder organization exports network configuration block information Participant organization uploads network configuration block information	In the founder's instance, download the network configuration block information (JSON file). Then in the participant's instance, upload the network configuration block.	Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service
Export and import the participant organization's peer nodes	Participant organization exports peers Founder organization imports peers	In the participant's instance, export the peers that you want to use on the network. Then in the founder's instance, import the peer nodes.	Export and Import Peer Nodes

Join Participant Organizations to the Channel and Set Anchor Peers

Task	Who Does This?	Description	More Information
Create a channel	Founder organization	In the founder's instance, create a channel that the founder and participants use to communicate. Add the founder's peers to the channel. You must select any newly added participants and assign them permissions on the channel. Note that instead of creating a new channel, you can add participants to an existing channel.	Create a Channel
Join participants to the channel	Participant organization	In the participant's instance, join the channel that was created in the founder's instance.	Join a Peer to a Channel
Set anchor peers on the founder and participants	Founder organization Participant organization	In the founder and participant instances, specify which peers you want to use as anchor peers. You must select at least one anchor peer for each member.	Add an Anchor Peer

Deploy the Chaincode Across the Blockchain Network

Task	Who Does This?	Description	More Information
Install the chaincode on the founder	Founder organization	In the founder's instance, upload and install the chaincode. Choose the peers to install the chaincode on.	Use Quick Deployment
Instantiate the chaincode and specify an endorsement policy on the founder	Founder organization	In the founder's instance, instantiate the chaincode to activate it on the network. An endorsement policy is required to specify the number of members that must approve chaincode transactions before they're submitted to the ledger.	Instantiate a Chaincode Specify an Endorsement Policy
Install the chaincode on the participant	Participant organization	In the participant's instance, install the chaincode that your network will use. Because you'll install the same chaincode that you installed and instantiated on the founder, you don't need to instantiate the chaincode on the participant. When the participant installs the chaincode, it's already instantiated.	Use Quick Deployment

Expose the Chaincode's REST API and Run Transactions

Task	Who Does This?	Description	More Information
Configure the founder's REST proxy node	Founder organization	In the founder's instance, modify the REST proxy node's attributes to specify the channel, chaincode, and peers that the network will use for transactions.	View and Edit the REST Proxy Node Configuration
Configure the participant's REST proxy node	Participant organization	In the participant's instance, modify the REST proxy node's attributes to specify the channel, chaincode, and peers that the network will use for transactions.	View and Edit the REST Proxy Node Configuration

Task	Who Does This?	Description	More Information
Invoke the chaincode and monitor network activity and ledger updates	Founder organization Participant organization	Begin using your network's chaincode for transactions. Both the founder and the participants can use their consoles to find out information about the activity on the network. Specifically, you can use the console's Channels tab to locate information about specific ledger transactions	Find Information About Nodes View a Channel's Ledger Activity

Join a Network

Participant organization are required to complete a wizard to join a blockchain network. The wizard displays the first time the participant organization opens its instance.

The wizard assists the participant organization with exporting the certificates to a JSON file to give to the network founder. The wizard also helps the participant import the founder's ordering service settings. For more information about the steps the founder and participant must complete to create a network, see Typical Workflow to Join a Participant Organization to an Oracle Blockchain Platform Network.

The participant's dashboard won't display until the wizard has been completed. If you're a network founder, then this wizard is never displayed.

1. Open the participant organization's console.

   The wizard that you'll use to join a network is displayed.

2. In the wizard, click **Export Certificates** and click the **Export** button.

   The Export dialog is displayed and includes the name of the JSON file the export will create.

3. Specify where to save the file. Click **OK** to save the certificates file.

4. Send the certificates JSON file to the network's founder. The network founder will import the participant's certificates file into the network.

5. Get the ordering services settings JSON file from the network founder. You'll import this file into your instance.

6. In the wizard, click **Import Ordering Service Settings**.

7. Click **Upload Ordering Service Settings**.

   The File Upload dialog is displayed.

8. In the File Upload dialog, browse for and select the JSON file containing the founder's ordering service settings information. Usually this file is named `<founderinstancename>-orderer-settings.json`. Click **Open**.

   The **Current Orderer Address** field updates with the address that Oracle Blockchain Platform extracted from the JSON file.

9. Click **Submit**.

   Your console's Dashboard tab is displayed.

Add Fabric Organizations to the Network

This topic contains information about joining Hyperledger Fabric organizations to an Oracle Blockchain Platform network.

Topics:

Typical Workflow to Join a Fabric Organization to an Oracle Blockchain Platform Network

Here are the tasks that a Fabric organization and the Oracle Blockchain Platform founder organization need to perform to join a Fabric organization to the Oracle Blockchain Platform network.

Task	Who Does This?	Description	More Information
Create the certificate file for the Fabric organization	Fabric organization	Find the Fabric organization’s Admin, CA, and TLS certificate information and use it to compose a JSON certificates file.	Create a Fabric Organization's Certificates File
Upload Fabric organization's certificate file to the Oracle Blockchain Platform network	Founder organization	Use the console to upload and import the Fabric organization's certificate file to add the Fabric organization to the network.	Import Certificates to Add Organizations to the Network
Create a channel	Founder organization	Create a new channel and add the Fabric organization to it.	Create a Channel
Export the ordering service settings from founder	Founder organization	Output the founder’s ordering services settings to a JSON file and send the file to the Fabric organization.	Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service

Task	Who Does This?	Description	More Information

Compose orderer certificate file	Fabric organization	Create a file named orderer.pem that includes the tlscacert information. Go to the exported ordering service settings file and copy the tlscacert information. After you paste the tlscacert information into the orderer.pem file, you must replace all instances of \n with the newline character.
		The orderer.pem file must have the following format:
		`-----BEGIN CERTIFICATE-----`
		`...`
		`...`
		`...`
		`-----END CERTIFICATE-----`
			Create a Fabric Organization's Certificates File

Provide ordering service settings	Founder organization	Open the ordering service settings file and find the ordering service's address and port and give them to the Fabric organization. For example:
		`"orderingServiceNodes": [`
		`{`
		`"address": "grpcs://example_address:7777",`
		`...`
		`}]`
			NA

Add the Fabric organization to the network	Fabric organization	The Fabric organization copies certificates into its environment, sets environment variables, fetches the genesis block, joins the channel, and installs the chaincode.	Prepare the Fabric Environment to Use the Oracle Blockchain Platform Network

Create a Fabric Organization's Certificates File

For a Fabric organization to join an Oracle Blockchain Platform network, it must write a certificates file containing its admincerts, cacerts, and tlscacerts information. The

Oracle Blockchain Platform founder organization imports this file to add the Fabric organization to the network.

The Fabric certificates information is stored in PEM files located in the Fabric organization’s MSP folder. For example, `network_name_example/crypto-config/peerOrganizations/example_org.com/msp/`.

The certificate file must be in written in JSON and contain the following fields:

- **mspid** — Specifies the name of the Fabric organization.
- **type** — Indicates that the organization is a network participant. This value must be `Participant`.
- **admincert** — Contains the contents of the organization’s Admin certificates file: `Admin@example_org.com-cert.pem`. When you copy the certificates information into the JSON file, you must replace each new line with `\n`.
- **cacert** — Contains the contents of the organization’s CA certificates file: `ca.example_org-cert.pem`. When you copy the certificates information into the JSON file, you must replace each new line with `\n`.
- **tlscert** — Contains the contents of the organization’s TLS certificate file: `tlsca.example_org-cert.pem`. When you copy the certificates information into the JSON file, you must replace each new line with `\n`.

This is how the file needs to be structured:

```
{
  "mspID": "examplemspID",
  "type":  "Participant",
  "certs": {
   "admincert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n",
   "cacert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n",
   "tlscacert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n"
 }
}
```

Prepare the Fabric Environment to Use the Oracle Blockchain Platform Network

You must modify the Fabric organization’s environment before it can use the Oracle Blockchain Platform network.

Confirm that the following prerequisite tasks were completed. For more information, see Typical Workflow to Join a Fabric Organization to an Oracle Blockchain Platform Network.

- The Fabric organization’s certificate file was created and sent to the Oracle Blockchain Platform network founder.

- The network founder uploaded the certificates file to add the Fabric organization to the network.
- The network founder created a new channel and added the Fabric organization to it.
- The network founder downloaded its ordering service settings and sent them to the Fabric organization.
- The Fabric organization created the orderer certificate file.
- The network founder gave the ordering service address and port to the Fabric organization.

You must add the Fabric organization and install and test the chaincode.

1. Navigate to the Fabric network directory and launch the peer container.
2. Fetch the channel's genesis block with this command:

   ```
   peer channel fetch 0 mychannel.block -o ${orderer_addr}:$
   {orderer_port} -c mychannel --tls --cafile orderer.pem --logging-
   level debug
   ```

   Where:

   - `{orderer_addr}` is the Founder's orderer address.
   - `{orderer_port}` is the Founder's port number.
   - `-c mychannel` is the name of the channel that the Founder created. This is the channel where the Fabric organization will send and receive transactions on the Oracle Blockchain Platform network.
   - `orderer.pem` is the Founder's orderer certificate file.
3. Join the channel with this command:

   ```
   peer channel join -b mychannel.block -o ${orderer_addr}:$
   {orderer_port} --tls --cafile orderer.pem --logging-level debug
   ```

4. Install the chaincode with this command:

   ```
   peer chaincode install -n mycc -v 1.0 -l "golang" -p ${CC_SRC_PATH}
   ```

   Where `CC_SRC_PATH` is the folder that contains the chaincode.
5. Instantiate the chaincode with this command:

   ```
   peer chaincode instantiate -o  ${orderer_addr}:${orderer_port} --
   tls --cafile orderer.pem -C mychannel -n mycc -l golang -v
   1.0 -c '{"Args":["init","a","100","b","200"]}' -P <policy_string> --
   logging-level debug
   ```

6. Invoke the chaincode with this command:

   ```
   peer chaincode invoke -o ${orderer_addr}:${orderer_port}  --tls
   true --cafile orderer.pem -C mychannel -n mycc -c '{"Args":
   ["invoke","a","b","10"]}' --logging-level debug
   ```

7. Query the chaincode with this command:

```
peer chaincode query -C mychannel -n mycc -c '{"Args":
["query","a"]}'  --logging-level debug
```

Add Organizations with Third-Party Certificates to the Network

This topic contains information about joining organizations using third-party certificates to an Oracle Blockchain Platform network.

Topics:

- Typical Workflow to Join an Organization with Third-Party Certificates to an Oracle Blockchain Platform Network
- Third-Party Certificate Requirements
- Create an Organization's Third-Party Certificates File
- Prepare the Third-Party Environment to Use the Oracle Blockchain Platform Network

Typical Workflow to Join an Organization with Third-Party Certificates to an Oracle Blockchain Platform Network

Organization with certificates issued by a third-party certificate authority (CA) can join the Oracle Blockchain Platform network as participants.

Client-only Organizations

These participants are client-only organizations and have no peers or orderers. They cannot create channels, join peers or install chaincode.

After joining the network, these organizations can use an SDK or a Hyperledger Fabric CLI to:

- Instantiate, invoke, and query chaincode if they're a client organization administrator.
- Invoke and query chaincode if they're a client organization non-administrator.

To control who can instantiate and invoke chaincode when client-only organizations are part of the network:

- The chaincode owner who installs the chaincode onto peers can decide who can instantiate the chaincode by using the Hyperledger Fabric `peer chaincode package -i` instantiation policy command to set the instantiation policy for the chaincode.
- The chaincode instantiator can use the Hyperledger Fabric `peer chaincode instantiate -P` endorsement policy command to set the endorsement policy controlling who can invoke the chaincode.
- The channel owner can decide who can invoke or query a chaincode by setting the channel proposal and query access control list. See Hyperledger Fabric Access Control Lists and What Are Channel Policies?.

Workflow

Here are the tasks that an organization with third-party certificates and the Oracle Blockchain Platform founder need to perform to join the organization to an Oracle Blockchain Platform network.

Task	Who Does This?	Description	More Information
Get the third-party certificates	Third-party certificates (participant) organization	Go to the third-party CA server and generate the required certificates files. Format the files as needed for import into the network.	Third-Party Certificate Requirements
Create the certificates file for import	Third-party certificates (participant) organization	Find the participant's Admin and CA certificate information and use it to compose a JSON certificates file.	Create an Organization's Third-Party Certificates File
Upload a certificate file for the third-party (participant) organization	Founder organization	Use the console to upload and import the participant's certificate file to add the participant to the network.	Import Certificates to Add Organizations to the Network

Export the ordering service settings from network founder and provide them to the third-party (participant) organization	Founder organization	Output the founder's ordering services settings to a JSON file and send the file to the participant. Open the ordering service settings file and find the ordering service's address and port and give them to the participant. For example:
		
		`"orderingServiceNodes": [`
		`{`
		`"address":`
		`"grpcs://example_address:7777"`
		`...`
		`}]`
			Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service

Create the channel	Founder	Create a new channel and add the participant to it.	Create a Channel
Install and instantiate the chaincode	Founder	In the founder's instance, upload, install, and instantiate the chaincode. Choose the network peers to install the chaincode on.	Use Quick Deployment

Task	Who Does This?	Description	More Information
Set up the third-party (participant) organization's environment	Third-party certificates (participant) organization	To query or invoke chaincodes, the participant must: • Add the founder's ordering service's address and port to the participant's environment. • Configure the environment to use Hyperledger Fabric CLI or SDKs. • Install the chaincode on peers.	Prepare the Third-Party Environment to Use the Oracle Blockchain Platform Network

Third-Party Certificate Requirements

To successfully join the network, an organization must generate the required third-party certificates. The information in these certificates is used to create the organization's certificates file, which is then imported into the founder's instance.

Which Certificates Do Organizations Need to Provide?

You must generate the following certificates from your CA server:

- Client Public Certificate
- CA Root Certificate

What Are the Requirements for These Certificates?

The certificates must meet the following requirements:

- When generating the private key, you must use the Elliptic Curve Digital Signature Algorithm (ECDSA). This algorithm is the only accepted algorithm for Fabric MSP keys.
- The Subject Key Identifier (SKI) is mandatory and you must indicate it as x509 extensions in the extension file.
- You must convert the key files from the .key to the .pem format.
- You must convert the certificates from the .crt to the .pem format.

Creating the Certificates

The following walkthrough is an example of how to use OpenSSL or the Hyperledger Fabric cryptogen utility to generate your certificates. For detailed information on the commands used, refer to:

- OpenSSL documentation
- cryptogen utility documentation

To create your certificates using OpenSSL:

1. Create a self-signed CA certificate/key:

   ```
   openssl ecparam -name prime256v1 -genkey -out ca.key
   openssl pkcs8 -topk8 -inform PEM -in ca.key -outform pem -nocrypt
   -out ca-key.pem
   openssl req -new -key ca-key.pem -out ca.csr
   openssl x509 -req -days 365 -in ca.csr -signkey ca-key.pem -out
   ca.crt -extensions x509_ext -extfile opensslca.conf
   openssl x509 -in ca.crt -out ca.pem -outform PEM
   ```

   Our example `opensslca.conf` file:

   ```
   [ req ]
   default_bits        = 2048
   distinguished_name  = subject
   req_extensions      = req_ext
   x509_extensions     = x509_ext
   string_mask         = utf8only


   [ subject ]
   countryName         = CN
   #countryName_default     = US


   stateOrProvinceName     = Beijing
   #stateOrProvinceName_default = NY

   localityName            = Beijing
   #localityName_default        = New York

   organizationName         = thirdpartyca, LLC
   #organizationName_default    = Example, LLC

   # Use a friendly name here because its presented to the user. The
   server's DNS
   #   names are placed in Subject Alternate Names. Plus, DNS names
   here is deprecated
   #   by both IETF and CA/Browser Forums. If you place a DNS name
   here, then you
   #   must include the DNS name in the SAN too (otherwise, Chrome and
   others that
   #   strictly follow the CA/Browser Baseline Requirements will fail).
   commonName          = thirdpartyca
   #commonName_default      = Example Company

   emailAddress            = ca@thirdpartyca.com



   # Section x509_ext is used when generating a self-signed
   certificate. I.e., openssl req -x509 ...
   [ x509_ext ]
   ```

```
subjectKeyIdentifier        = hash
authorityKeyIdentifier  = keyid,issuer

# You only need digitalSignature below. *If* you don't allow
#   RSA Key transport (i.e., you use ephemeral cipher suites), then
#   omit keyEncipherment because that's key transport.
basicConstraints        = CA:TRUE
keyUsage            = Certificate Sign, CRL Sign, digitalSignature,
keyEncipherment
subjectAltName          = @alternate_names
nsComment           = "OpenSSL Generated Certificate"

# RFC 5280, Section 4.2.1.12 makes EKU optional
#   CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me
confused
#   In either case, you probably only need serverAuth.
# extendedKeyUsage  = serverAuth, clientAuth

# Section req_ext is used when generating a certificate signing
request. I.e., openssl req ...
[ req_ext ]

subjectKeyIdentifier        = hash

basicConstraints        = CA:FALSE
keyUsage            = digitalSignature, keyEncipherment
subjectAltName          = @alternate_names
nsComment           = "OpenSSL Generated Certificate"

# RFC 5280, Section 4.2.1.12 makes EKU optional
#   CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me
confused
#   In either case, you probably only need serverAuth.
# extendedKeyUsage  = serverAuth, clientAuth

[ alternate_names ]

DNS.1       = localhost
DNS.2       = thirdpartyca.com
#DNS.3       = mail.example.com
#DNS.4       = ftp.example.com

# Add these if you need them. But usually you don't want them or
#   need them in production. You may need them for development.
# DNS.5       = localhost
# DNS.6       = localhost.localdomain
# DNS.7       = 127.0.0.1
```

2. Create a user certificate/key using above CA key:

```
openssl ecparam -name prime256v1 -genkey -out user.key
openssl pkcs8 -topk8 -inform PEM -in user.key -outform pem -nocrypt
-out user-key.pem
openssl req -new -key user-key.pem -out user.csr
openssl x509 -req -days 365 -sha256 -CA ca.pem -CAkey ca-key.pem
-CAserial ca.srl -CAcreateserial -in user.csr -out user.crt -
extensions x509_ext -extfile openssl.conf
openssl x509 -in user.crt -out user.pem -outform PEM
```

Our example `openssl.conf` file:

```
[ req ]
default_bits        = 2048
default_keyfile     = tls-key.pem
distinguished_name  = subject
req_extensions      = req_ext
x509_extensions     = x509_ext
string_mask         = utf8only

# The Subject DN can be formed using X501 or RFC 4514 (see RFC 4519
for a description).
# Its sort of a mashup. For example, RFC 4514 does not provide
emailAddress.
[ subject ]
countryName         = CN
#countryName_default     = US

stateOrProvinceName     = Beijing
#stateOrProvinceName_default = NY

localityName            = Beijing
#localityName_default        = New York

organizationName         = thirdpartyca, LLC
#organizationName_default    = Example, LLC

# Use a friendly name here because its presented to the user. The
server's DNS
#   names are placed in Subject Alternate Names. Plus, DNS names
here is deprecated
#   by both IETF and CA/Browser Forums. If you place a DNS name
here, then you
#   must include the DNS name in the SAN too (otherwise, Chrome and
others that
#   strictly follow the CA/Browser Baseline Requirements will fail).
commonName          = admin@thirdpartyca.com
#commonName_default      = Example Company

emailAddress            = admin@thirdpartyca.com
#emailAddress_default        = test@example.com

# Section x509_ext is used when generating a self-signed
certificate. I.e., openssl req -x509 ...
[ x509_ext ]
subjectKeyIdentifier        = hash
authorityKeyIdentifier  = keyid,issuer

# You only need digitalSignature below. *If* you don't allow
#   RSA Key transport (i.e., you use ephemeral cipher suites), then
#   omit keyEncipherment because that's key transport.
basicConstraints        = CA:FALSE
keyUsage            = digitalSignature, keyEncipherment

subjectAltName          = @alternate_names
nsComment           = "OpenSSL Generated Certificate"

# RFC 5280, Section 4.2.1.12 makes EKU optional
#   CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me 
confused
#   In either case, you probably only need serverAuth.
#extendedKeyUsage  = Any Extended Key Usage
#extendedKeyUsage = serverAuth, clientAuth


# Section req_ext is used when generating a certificate signing 
request. I.e., openssl req ...
[ x509_ca_ext ]
subjectKeyIdentifier        = hash
authorityKeyIdentifier  = keyid,issuer


# You only need digitalSignature below. *If* you don't allow
#   RSA Key transport (i.e., you use ephemeral cipher suites), then
#   omit keyEncipherment because that's key transport.
basicConstraints        = CA:TRUE
keyUsage            = Certificate Sign, CRL Sign, digitalSignature, 
keyEncipherment
subjectAltName          = @alternate_names
nsComment           = "OpenSSL Generated Certificate"


# RFC 5280, Section 4.2.1.12 makes EKU optional
#   CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me 
confused
#   In either case, you probably only need serverAuth.
#extendedKeyUsage  = Any Extended Key Usage
extendedKeyUsage = serverAuth, clientAuth


# Section req_ext is used when generating a certificate signing 
request. I.e., openssl req ...
[ req_ext ]
subjectKeyIdentifier        = hash
basicConstraints        = CA:FALSE
keyUsage            = digitalSignature, keyEncipherment
subjectAltName          = @alternate_names
nsComment           = "OpenSSL Generated Certificate"


# RFC 5280, Section 4.2.1.12 makes EKU optional
#   CA/Browser Baseline Requirements, Appendix (B)(3)(G) makes me 
confused
#   In either case, you probably only need serverAuth.
```

```
#extendedKeyUsage  = Any Extended Key Usage
#extendedKeyUsage = serverAuth, clientAuth

[ alternate_names ]
DNS.1       = localhost
DNS.3       = 127.0.0.1
DNS.4       = 0.0.0.0
# Add these if you need them. But usually you don't want them or
#   need them in production. You may need them for development.
# DNS.5       = localhost
# DNS.6       = localhost.localdomain
# DNS.7       = 127.0.0.1
# IPv6 localhost
# DNS.8     = ::1
```

To create your certificates using the Hyperledger Fabric cryptogen utility:

- The following cryptogen commands are used to create Hyperledger Fabric key material:

  ```
  cryptogen generate --config=./crypto-config.yaml
  ```

  Our example `crypto-config.yaml` file:

  ```
  # Copyright IBM Corp. All Rights Reserved.
  #
  # SPDX-License-Identifier: Apache-2.0
  #

  #
  --------------------------------------------------------------------
  -------
  # "PeerOrgs" - Definition of organizations managing peer nodes
  #
  --------------------------------------------------------------------
  -------
  PeerOrgs:
    #
  --------------------------------------------------------------------
  -------
    # Org1
    #
  --------------------------------------------------------------------
  -------
    - Name: Org1
      Domain: org1.example.com
      EnableNodeOUs: true
      #
  --------------------------------------------------------------------
  -------
      # "Specs"
      #
  --------------------------------------------------------------------
  -------
  ```

```
    # Uncomment this section to enable the explicit definition of
hosts in your
    # configuration.  Most users will want to use Template, below
    #
    # Specs is an array of Spec entries.  Each Spec entry consists
of two fields:
    #   - Hostname:   (Required) The desired hostname, sans the
domain.
    #   - CommonName: (Optional) Specifies the template or explicit
override for
    #                 the CN.  By default, this is the template:
    #
    #                              "{{.Hostname}}.{{.Domain}}"
    #
    #                 which obtains its values from the
Spec.Hostname and
    #                 Org.Domain, respectively.
    #
--------------------------------------------------------------------
-------
    # Specs:
    #   - Hostname: foo # implicitly "foo.org1.example.com"
    #     CommonName: foo27.org5.example.com # overrides Hostname-
based FQDN set above
    #   - Hostname: bar
    #   - Hostname: baz
    #
--------------------------------------------------------------------
-------
    # "Template"
    #
--------------------------------------------------------------------
-------
    # Allows for the definition of 1 or more hosts that are created
sequentially
    # from a template. By default, this looks like "peer%d" from 0
to Count-1.
    # You may override the number of nodes (Count), the starting
index (Start)
    # or the template used to construct the name (Hostname).
    #
    # Note: Template and Specs are not mutually exclusive.  You may
define both
    # sections and the aggregate nodes will be created for you.
Take care with
    # name collisions
    #
--------------------------------------------------------------------
-------
    Template:
      Count: 2
      # Start: 5
      # Hostname: {{.Prefix}}{{.Index}} # default
    #
--------------------------------------------------------------------
```

```
-------
    # "Users"
    #
--------------------------------------------------------------------
-------
    # Count: The number of user accounts _in addition_ to Admin
    #
--------------------------------------------------------------------
-------
    Users:
      Count: 1
  #
--------------------------------------------------------------------
-------
  # Org2: See "Org1" for full specification
  #
--------------------------------------------------------------------
-------
  - Name: Org2
    Domain: org2.example.com
    EnableNodeOUs: true
    Template:
      Count: 2
    Users:
      Count: 1
```

What's Next?

After confirming that you've outputted and updated the proper files, you can then create the certificates file for import into the Oracle Blockchain Platform network. See Create an Organization's Third-Party Certificates File.

Create an Organization's Third-Party Certificates File

To join an Oracle Blockchain Platform network, the organization must write a certificates file containing its admincert and cacert information. The network founder imports this file to add the organization to the network.

Go to the certificates files that you generated from the CA server to find the information that you need to create the certificates file. See Third-Party Certificate Requirements.

The certificates file must be in written in JSON and contain the following fields:

- **mspid** — Specifies the name of the organization.
- **type** — Indicates that the organization is a network participant. This value must be `Participant`.
- **admincert** — Contains the contents of the organization's Admin certificates file. When you copy the certificates information into the JSON file, you must replace each new line with `\n`.
- **cacert** — Contains the contents of the organization's CA certificates file. When you copy the certificates information into the JSON file, you must replace each new line with `\n`.

This is how the file needs to be structured:

```
{
  "mspID": "examplemspID",
  "type":  "Participant",
  "certs": {
   "admincert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n",
   "cacert": "-----BEGIN CERTIFICATE-----
\nexample_certificate\nexample_certificate==\n-----END CERTIFICATE-----
\n"
 }
}
```

Prepare the Third-Party Environment to Use the Oracle Blockchain Platform Network

You must set up the third-party organization's environment before it can use the Oracle Blockchain Platform network.

Confirm that the following prerequisite tasks were completed. For information, see Typical Workflow to Join an Organization with Third-Party Certificates to an Oracle Blockchain Platform Network.

- The third-party organization’s certificate file was created and sent to the Oracle Blockchain Platform network founder.
- The network founder uploaded the certificates file to add the third-party organization to the network.
- The network founder exported the orderer service's settings and gave the service's address and port to the third-party organization and the organization added them to the environment.
- The network founder created a new channel and added the third-party organization to it.
- The network founder installed and instantiated the chaincode.

Setup organization's Environment

Before the third-party organization can successfully use the Oracle Blockchain Platform network, it must set up its environment to use Hyperledger Fabric CLI or SDKs. See Welcome to Hyperledger Fabric.

Install the Chaincode

The third-party organization must install the chaincode on the peers. These peers must then be joined to the channel so that the chaincode can be invoked.

Instantiate the Chaincode

If needed, the third-party organizations can instantiate the chaincode on the channel. For example:

```
export  CORE_PEER_TLS_ENABLED=true
export  CORE_PEER_TLS_ROOTCERT_FILE=$PWD/tls-ca.pem
export  CORE_PEER_MSPCONFIGPATH=$PWD/crypto-config/peerOrganizations/
customerorg1.com/users/Admin@customerorg1.com/msp
export  CORE_PEER_LOCALMSPID="customerorg1"

### gets channel name from input###
CHANNEL_NAME=$1

echo "######### going to instantiate chaincode on channel $
{CHANNEL_NAME} ##########"
CORE_PEER_ADDRESS=${peer_host}:${port} peer chaincode instantiate
-o ${peer_host}:${port}  --tls $CORE_PEER_TLS_ENABLED --cafile
./tls-ca.pem -C ${CHANNEL_NAME}  -n obcs-example02 -v v0 -c '{"Args":
["init","a","100","b","200"]}'
```

Invoke the Chaincode

Third-party organizations use the Hyperledger Fabric CLI or SDKs to invoke the chaincode. For example:

```
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_TLS_ROOTCERT_FILE=$PWD/tls-ca.pem
export CORE_PEER_MSPCONFIGPATH=$PWD/crypto-config/peerOrganizations/
customerorg1.com/users/User1@customerorg1.com/msp
export CORE_PEER_LOCALMSPID="customerorg1"

### gets channel name from input ###
CHANNEL_NAME=$1

#### do query or invoke on chaincode ####

CORE_PEER_ADDRESS=${peer_host}:${port} peer chaincode query -C
${CHANNEL_NAME} -n $2 -c '{"Args":["query","a"]}'

CORE_PEER_ADDRESS=${peer_host}:${port} peer chaincode invoke -o
${peer_host}:${port} --tls $CORE_PEER_TLS_ENABLED --cafile ./tls-
ca.pem -C ${CHANNEL_NAME} -n $2 -c '{"Args":["invoke","a","b","10"]}'
```

6

Develop Chaincodes

This topic contains information to help you understand how to write and test chaincodes for use in Oracle Blockchain Platform.

Topics

Write a Chaincode

A chaincode is written in Go, Node.js, or Java and then packaged into a ZIP file that is installed on the Oracle Blockchain Platform network.

Chaincodes define the data schema in the ledger, initialize it, perform updates when triggered by applications, and respond to queries. Chaincodes can also post events that allow applications to be notified and perform downstream operations. For example, after purchase orders, invoices, and delivery records have been matched by a chaincode, it can post an event so that a subscribing application can process related payments and update an internal ERP system.

Resources for Chaincode Development

Oracle Blockchain Platform uses Hyperledger Fabric as its foundation. Use the Hyperledger Fabric documentation to help you write valid chaincodes.

- Welcome to Hyperledger Fabric. The Key Concepts and Tutorials sections should be read before you write you own chaincode.
- Go Programming Language. The Go compilers, tools, and libraries provide a variety of resources that simplify writing chaincodes.
- Package shim. Package shim provides APIs for the chaincode to access its state variables, transaction context and call other chaincodes. This documents the actual syntax required for your chaincode.

Oracle Blockchain Platform provides downloadable samples that help you understand how to write chaincodes and applications. See What Are Chaincode Samples?

You can add rich-query syntax to your chaincodes to query the state database. See SQL Rich Query Syntax and CouchDB Rich Query Syntax.

Package and Zip a Go Chaincode

Once you've written your chaincode, place it in a ZIP file. You don't need to create a package for the Go chaincode or sign it — the Oracle Blockchain Platform install and instantiation process does this for you as described in Typical Workflow to Deploy Chaincodes.

If your chaincode has any external dependencies, you can place them in the vendor directory of your ZIP file.

Package and Zip a Node.js Chaincode

If you're writing a Node.js chaincode, you need to create a `package.json` file with two sections:

- The scripts section declares how to launch the chaincode.
- The dependencies section specifies the dependencies.

The following is a sample `package.json` for a Node.js chaincode:

```
{
    "name": "chaincode_example02",
    "version": "1.0.0",
    "description": "chaincode_example02 chaincode implemented in 
Node.js",
    "engines": {
        "node": ">=8.4.0",
        "npm": ">=5.3.0"
    },
    "scripts": { "start" : "node chaincode_example02.js" },
    "engine-strict": true,
    "license": "Apache-2.0",
    "dependencies": {
        "fabric-shim": "~1.3.0"
    }
}
```

The packaging rules for a Node.js chaincode are:

- `package.json` must be in the root directory.
- The entry JavaScript file can be located anywhere in the package.
- If `"start" : "node <start>.js"` isn't specified in the `package.json`, `server.js` must be in the root directory.

Place the chaincode and package file in a zip file to install it on Oracle Blockchain Platform.

Package and Zip a Java Chaincode

If you're writing a Java chaincode, you can choose Gradle or Maven to build the chaincode.

If you're using Gradle, place the chaincode, build.gradle, and settings.gradle in a zip file to install it on Oracle Blockchain Platform. The following is a sample file list of a chaincode zip package:

```
Archive:  example_gradle.zip
 Length      Date    Time    Name
---------  ---------- -----   ----
      610  02-14-2019 01:36   build.gradle
       54  02-14-2019 01:28   settings.gradle
        0  02-14-2019 01:28   src/
        0  02-14-2019 01:28   src/main/
        0  02-14-2019 01:28   src/main/java/
        0  02-14-2019 01:28   src/main/java/org/
        0  02-14-2019 01:28   src/main/java/org/hyperledger/
        0  02-14-2019 01:28   src/main/java/org/hyperledger/fabric/
        0  02-14-2019 01:28   src/main/java/org/hyperledger/fabric/example/
     5357  02-14-2019 01:28   src/main/java/org/hyperledger/fabric/example/
SimpleChaincode.java
---------                     -------
     6021                     10 files
```

If you're using Maven, place the chaincode and pom.xml in a zip file to install it on Oracle Blockchain Platform. The following is a sample file list of a chaincode zip package:

```
Archive:  example_maven.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     3313  02-14-2019 01:52   pom.xml
        0  02-14-2019 01:28   src/
        0  02-14-2019 01:28   src/chaincode/
        0  02-14-2019 01:28   src/chaincode/example/
     4281  02-14-2019 01:28   src/chaincode/example/SimpleChaincode.java
---------                     -------
     7594                     5 files
```

Testing a Chaincode

After you write your chaincode, then you need to test it. See:

- Use a Mock Shim to Test a Chaincode
- Deploy a Chaincode on a Peer to Test the Chaincode

Installing and Instantiating a Chaincode

Once you’ve tested your chaincode, you can deploy it following the information in Typical Workflow to Deploy Chaincodes.

Upgrading a Chaincode

A chaincode may be upgraded any time by changing its version. The chaincode name must be the same or it would be considered a totally different chaincode.

1. Change the chaincode version
2. Follow the instructions in Upgrade a Chaincode to install and upgrade the new version of the chaincode.

Use a Mock Shim to Test a Chaincode

This method of testing involves using a mock version of the stub `shim.ChaincodeStubInterface`. With this you can simulate some functionality of your chaincode before deploying it to Oracle Blockchain Platform. You can also use this library to build unit tests for your chaincode.

Manually Vendor the Shim with a Chaincode

This applies to Oracle Blockchain Platform 19.1.1, 19.1.3, 19.2.1, and 19.2.3.

In Hyperledger Fabric, the fabric-ccenv image contains the `github.com/hyperledger/fabric/core/chaincode/shim` (shim) package. This allows you to package a chaincode without needing to include the shim. However, this may cause issues in future Hyperledger Fabric releases, and it may cause issues when using packages that are included with the shim.

Workaround: To avoid potential issues, you should manually vendor the shim package with the chaincode prior to using the `peer` command-line interface for packaging and installing a chaincode, or packaging or installing a chaincode. See https://jira.hyperledger.org/browse/FAB-5177.

Use a Mock Shim to Test a Chaincode

1. Create a test file that matches the name of the chaincode file.

   For example, if `car_dealer.go` is the actual implementation code for you smart contract, you would create a test suite called `car_dealer_test.go` containing all the tests for `car_dealer.go`. The test suite filename should be in the `*_test.go` format.

2. Create your package and import statements.

   ```
   package main

   import (
       "fmt"
       "testing"

       "github.com/hyperledger/fabric/core/chaincode/shim"
   )
   ```

3. Create your unit test.

   ```
   /*
   * TestInvokeInitVehiclePart simulates an initVehiclePart
   transaction on the CarDemo cahincode
    */
   func TestInvokeInitVehiclePart(t *testing.T) {
       fmt.Println("Entering TestInvokeInitVehiclePart")

       // Instantiate mockStub using CarDemo as the target chaincode
   to unit test
       stub := shim.NewMockStub("mockStub", new(CarDemo))
       if stub == nil {
           t.Fatalf("MockStub creation failed")
       }

       var serialNumber = "ser1234"

       // Here we perform a "mock invoke" to invoke the function
   "initVehiclePart" method with associated parameters
       // The first parameter is the function we are invoking
       result := stub.MockInvoke("001",
           [][]byte{[]byte("initVehiclePart"),
               []byte(serialNumber),
               []byte("tata"),
               []byte("1502688979"),
               []byte("airbag 2020"),
               []byte("aaimler ag / mercedes")})
   ```

```
    // We expect a shim.ok if all goes well
    if result.Status != shim.OK {
        t.Fatalf("Expected unauthorized user error to be returned")
    }

    // here we validate we can retrieve the vehiclePart object we 
just committed by serianNumber
    valAsbytes, err := stub.GetState(serialNumber)
    if err != nil {
        t.Errorf("Failed to get state for " + serialNumber)
    } else if valAsbytes == nil {
        t.Errorf("Vehicle part does not exist: " + serialNumber)
    }
}
```

Note:

Not all interfaces of the stub are implemented. Stub functions

- `GetQueryResult`
- `GetHistoryForKey`

are not supported, and attempting to call either of these will result in an error.

Deploy a Chaincode on a Peer to Test the Chaincode

After you create a chaincode, you must install, instantiate, and invoke it to test that it works correctly.

If you need help writing a chaincode, see Write a Chaincode.

Follow these steps to deploy and test your chaincode.

1. Identify the channel or create a new channel and add peers to it. See Join a Peer to a Channel.
2. Install the chaincode on the peers and instantiate it on the channel. See Use Quick Deployment.
3. Use the *Invoke* and *query* REST APIs to test the chaincode with cURL through the REST proxy. See REST API for Oracle Blockchain Platform for descriptions of each endpoint and correct cURL syntax to invoke each operation.
4. Go to the Channels tab in the console and locate and click the name of the channel running the blockchain.
5. In the channel’s **Ledger** pane, view the chaincode’s ledger summary.

7 Deploy and Manage Chaincodes

This topic contains information to help you deploy (install, instantiate, upgrade, and enable in the REST proxy), monitor, and find information about the chaincodes on the network.

Topics

Typical Workflow to Deploy Chaincodes

Here are the common tasks for deploying chaincodes.

You must be an administrator to perform these tasks.

Task	Description	More Information
Use the wizard to fully or partially deploy a chaincode	For testing, use Quick Deployment to perform the deployment in one step, using default settings. For production, use Advanced Deployment to specify the deployment settings such as which peers to install the chaincode on and the endorsement policy you want to use. With Advanced Deployment you can instantiate the chaincode and enable it in the REST proxy now or later.	Use Quick Deployment Use Advanced Deployment

Task	Description	More Information
Instantiate a chaincode	Instantiate the chaincode after you've installed it.	Instantiate a Chaincode
Upgrade the chaincode	Upload and instantiate a newer version of a chaincode, or pick an older version of the chaincode to use.	Upgrade a Chaincode

Use Quick Deployment

Use the quick deployment option to perform a one-step chaincode deployment. This option is recommended for chaincode testing.

The quick deployment uses default settings, installs the chaincode on all peers in the channel, instantiates the chaincode using the default endorsement policy, and enables the chaincode in the REST proxy.

Note the following information:

- The process to deploy sample chaincodes is different than the process described in this topic. See Explore Oracle Blockchain Platform Using Samples.
- You can use the advanced deployment option to put your chaincode into production on the network. See Use Advanced Deployment.
- You can't delete a chaincode from the network.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. In the Chaincodes tab, click **Deploy a New Chaincode**.

   The Deploy Chaincode page is displayed.
3. Click **Quick Deployment**.

   The Deploy Chaincode (Quick) page is displayed.
4. In the **Chaincode Name** field, enter a unique name for the chaincode. In the **Version** field enter a string value to specify the chaincode's version number.

   The Oracle Blockchain Platform chaincode name and version requirements are different than the Hyperledger Fabric requirements. You must use the Oracle Blockchain Platform naming requirements. Use these guidelines when naming the chaincode:

   - Use ASCII alphanumeric characters, (") quotes, dashes (-), and underscores (_).
   - The name must start and end only with ASCII alphanumeric characters. For example, you can't use names like *_mychaincode* or *mychaincode_*.
   - Dashes (-) and underscores (_) must be followed with ASCII alphanumeric characters. For example, you can't use names like *my--chaincode* or *my-_chaincode*.
   - The name must be 1 to 64 characters long.
   - A chaincode version can contain a period (.).
5. Review the other default settings and modify them as needed.

6. Click the **Chaincode Source** field and browse for the chaincode ZIP file to upload and deploy.
7. Click **Submit**.

   The chaincode is installed on the channel’s peers, instantiated, and enabled in the REST proxy. The deployed chaincode’s name is displayed in the Chaincode tab’s table.

Use Advanced Deployment

Use the advanced deployment option to specify the parameters required to deploy a chaincode into a production environment. For example, you’ll specify which peers to install the chaincode on and the endorsement policy to use.

With the advanced deployment wizard, you’ll install the chaincode on the peers you select.

Note the following information:

- The process to deploy sample chaincodes is different than the process described in this topic. See Explore Oracle Blockchain Platform Using Samples.
- You can use the quick deployment option for chaincode testing. Quick deployment is a one-step deployment that uses default settings, installs the chaincode on all peers in the channel, and instantiates the chaincode using a default endorsement policy. See Use Quick Deployment.
- You can’t delete a chaincode from the network.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. In the Chaincodes tab, click **Deploy a New Chaincode**.

   The Deploy Chaincode page is displayed.
3. Click **Advanced Deployment**.

   The Deploy Chaincode (Advanced) Step 1 of 3: Install page is displayed.
4. In the **Chaincode Name** field, enter a unique name for the chaincode. In the **Version** field, enter the chaincode’s version number.

   The Oracle Blockchain Platform chaincode name and version requirements are different than the Hyperledger Fabric requirements. You must use the Oracle Blockchain Platform naming requirements. Use these guidelines when naming the chaincode:

   - Use ASCII alphanumeric characters, (") quotes, dashes (-), and underscores (_).
   - The name must start and end only with ASCII alphanumeric characters. For example, you can't use names like *_mychaincode* or *mychaincode_*.
   - Dashes (-) and underscores (_) must be followed with ASCII alphanumeric characters. For example, you can't use names like *my--chaincode* or *my-_chaincode*.
   - The name must be 1 to 64 characters long.
   - A chaincode version can contain a period (.).

5. Select one or more network peers to install the chaincode onto. To provide high availability, Oracle suggests that you choose the appropriate number of peers from each partition. Also, the peers you choose must be joined to the channel that you'll instantiate the chaincode on.
6. Click the **Chaincode Source** field and browse for the chaincode ZIP file to upload and deploy. Click **Next**.

   The chaincode is installed and the Deploy Chaincode (Advanced) Step 2 of 3: Instantiate page is displayed.
7. Decide if you want to instantiate the chaincode now or later.
   - Click **Close** to close the wizard and instantiate later.
   - To instantiate now, select the channel to instantiate the chaincode on and the peers to instantiate the chaincode to. If required, enter initial parameters, an endorsement policy, transient map, and private data collections. Note the following information:
     – Instantiation compiles, builds, and initializes the chaincode on the peers.
     – If you leave the endorsement policy blank, then Oracle Blockchain Platform uses the default endorsement policy. The default endorsement policy gets an endorsement from any peer on the network.
     – When instantiation is complete, the peers are able to accept chaincode invocations and can endorse transactions.

     Click **Next**.

   The chaincode is instantiated.

Update REST Proxy Settings for Running Chaincodes

If you're using Node.js or Java chaincodes which are dependent on external libraries, a few proxy settings must be updated.

You must complete these steps before you instantiate the chaincode. See Instantiate a Chaincode.

Node.js Chaincode

When your Node.js chaincode is instantiated, npm is used to install all dependency libraries from the internet, so in an Oracle Blockchain Platform instance, if there is such a dependancy, you need to ensure the `bcs/fabric-ccenv` image has internet access.

Set the HTTP proxy for the `bcs/fabric-ccenv` image following these steps:

1. Create a Docker file in any location in your VM with the following content (where http://hostname:port is your HTTP proxy access entry):

```
FROM bcs/fabric-ccenv:latest
ENV npm_config_proxy http://hostname:port
```

2. Build the image again:

```
docker build -f Dockerfile -t bcs/fabric-ccenv:latest
```

Java Chaincode

To configure the proxy to run Java chaincode:

1. Gradle-only: Create `gradle.properties` in a local directory, and add the following content in it:

   ```
   systemProp.http.proxyHost=[proxy host]
   systemProp.http.proxyPort=[proxy port]
   systemProp.https.proxyHost=[proxy host]
   systemProp.https.proxyPort=[proxy port]
   ```

2. Maven-only: Create `settings.xml` in a local directory, and add the following content in it:

   ```
   <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
   https://maven.apache.org/xsd/settings-1.0.0.xsd">
   <localRepository/>
   <interactiveMode/>
   <usePluginRegistry/>
   <offline/>
   <pluginGroups/>
   <servers/>
   <mirrors/>
   <proxies>
   <proxy>
   <id>httpproxy</id>
   <active>true</active>
   <protocol>http</protocol>
   <host>[proxy host]</host>
   <port>[proxy port]</port>
   </proxy>
   <proxy>
   <id>httpsproxy</id>
   <active>true</active>
   <protocol>https</protocol>
   <host>[proxy host]</host>
   <port>[proxy port]</port>
   </proxy>
   </proxies>
   <profiles/>
   <activeProfiles/>
   </settings>
   ```

3. Create a Docker file and add the following to it:

   ```
   FROM bcs/fabric-javaenv:latest
   COPY gradle.properties /root/.gradle/
   COPY settings.xml /root/.m2/
   ```

4. Create a new image:

```
docker build -f dockerfile -t bcs/fabric-javaenv:latest
```

Instantiate a Chaincode

Instantiating a chaincode compiles, builds, and initializes the chaincode on the peers where the chaincode is installed. When instantiation is complete, the peers are able to accept chaincode invocations and can endorse transactions.

Note the following information:

- You must install the chaincode on the required peers before you can instantiate it.
- If you're working on a channel that contains multiple members and have instantiated the chaincode on one member, then you don’t have to instantiate the chaincode on the other members where you installed the same chaincode. In such cases, the chaincode is already instantiated and running on all members on the channel.
- You can instantiate more than one chaincode on a channel.
- The process to instantiate the sample chaincodes is different than the instantiation process described in this topic. See Explore Oracle Blockchain Platform Using Samples.
- After you instantiate the chaincode, then you can optionally enable it in the REST proxy.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. In the Chaincodes tab, click the arrow to expand the chaincode’s version list.
3. Locate the chaincode version and click its **More Actions** menu, and select **Instantiate**.

   The Instantiate Chaincode dialog is displayed.
4. Enter information about where and how to instantiate the chaincode.

Field	Description
Channel	Select the channel for the chaincode to run on.
Peers	Select the peer or peers you want to use the chaincode. This list shows the peers that you installed the chaincode onto.
Initial Parameter	Enter the input parameters that you want to pass to the chaincode. Go to the chaincode to find the initial parameters values.
Endorsement Policy	In this section, specify the number and role of members required to endorse the chaincode. If you don’t specify an endorsement policy, then the default endorsement policy is used. The default endorsement policy gets an endorsement from any peer on the network.

Field	Description
Transient Map	The data that is passed into the chaincode is the transaction payload and the transient map. The transaction payload is recorded in the ledger and is visible to anyone who can access the ledger through the query system chaincode. Use a transient map to pass private data such as keys that you don't want stored in the ledger. In this section, provide the required keys and values. The information you provide is maintained on the peer node and is sent to the chaincode when a transaction is executed. If you're adding private data collections, then specify a transient map to pass the private data from the client to the peers for endorsement.
(New in v19.1.3) Private Data Collections	In this section, add one or more private data collections. Private data collections specify subsets of organizations that endorse, commit, or query private data on the channel you instantiate the chaincode on.

5. Click **Instantiate**.

   The chaincode is instantiated.

6. To confirm that the chaincode was instantiated, go to the Channels tab and click the name of the channel that you instantiated the chaincode on. Go to the Instantiated Chaincodes tab and confirm that the chaincode is listed in the summary table.

Specify an Endorsement Policy

You can add an endorsement policy when you instantiate a chaincode. An endorsement policy specifies the members with peers that must approve, or properly endorse, a chaincode transaction before it's added to a block and submitted to the ledger.

Endorsement guarantees the legitimacy of a transaction. When you instantiate a chaincode on a channel, you can specify an endorsement policy. If you don't specify an endorsement policy, then the default endorsement policy is used. The default endorsement policy gets an endorsement from any peer on the network.

A member's endorsing peers must have ReaderWriter permissions on the channel. When a transaction is processed, each endorsing peer returns a signed read-write set. After the client has enough endorsements to meet the endorsement policy requirements, then the client bundles the common read-write set with the signature from the endorsing peers and sends everything to the ordering service, which orders and commits the transactions into blocks and then to the ledger.

You can go to the Channels tab to view an instantiated chaincode's endorsement policy. See View an Endorsement Policy. You can't modify an instantiated chaincode's endorsement policy. If you need to change an endorsement policy, then you must reinstantiate the chaincode or upgrade it to another version and specify a different endorsement policy.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.
2. Locate the chaincode that you want to instantiate and begin the instantiation process.
3. Expand the Endorsement Policy section. Click **Add Identity** to add members to the policy as needed.

Field	Description
MSP ID	From the dropdown menu, select the endorser peer’s organization.
Role	Select the corresponding peer role required by the endorsement policy. Usually this will be member. You can find a peer’s role by viewing its configuration information.
Policy Expression Mode	In most cases, you’ll use **Basic**. Select **Advanced** to provide an expression string. See the Hyperledger Fabric documentation for information about how to write a valid expression string.
Signed By	Select how many members with endorsing peers (peers with ReaderWriter permissions) on the channel must endorse the chaincode transactions to make them valid.

4. Complete the other fields on the Instantiate Chaincode page as needed.
5. Click **Instantiate**.

View an Endorsement Policy

You can view an instantiated chaincode's endorsement policy.

You might need to view an instantiated chaincode's endorsement policy to see how it was set up, how you need to choose transaction endorsers based on the policy, or to help resolve an endorsement failure.

You can’t modify the endorsement policy for an instantiated chaincode. If you need to change an endorsement policy, then you must reinstantiate the chaincode or upgrade it to another version and specify a different endorsement policy.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the table lists the chaincodes installed on the network.
2. Locate the chaincode that you want to view endorsement policy information for and expand it in the table.
3. Click the chaincode version that you want.

   The Chaincode Version Information page is displayed.
4. In the Instantiated on Channels tab, locate the channel that you want, click **More Actions**, and select **View Endorsement Policy**.

   The Chaincode Endorsement Policy page is displayed.

Find Information About Chaincodes

You can find information about the chaincodes in your network. For example, how many peers the chaincode is installed on and if the chaincode has been instantiated.

You can’t delete a chaincode from the network.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the chaincode table lists the chaincodes and versions installed on the network.
2. In the chaincode table, locate the chaincode that you want information for and expand it to see information about its versions, path, how many peers it’s installed on, and how many channels it’s instantiated on.

Note the following information:

- When you stop a peer node, Oracle Blockchain Platform removes the peer's listing on the Chaincodes tab.
- If you stop all peers that have the chaincode installed, then the Chaincodes tab doesn't list the chaincode. To list the chaincode, start at least one peer node that has the chaincode installed on it.

3. Use the chaincode table as a starting point to perform chaincode-related tasks, such as instantiate, enable it in the REST proxy, and upgrade to a new version.

Manage Chaincode Versions

Each chaincode that you install or upgrade has a version number. Once installed, a chaincode and any of its versions can't be deleted.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the chaincode table lists the chaincodes installed on the network.
2. Locate the chaincode that you want version information for and expand it to see a list of versions.
3. Click a version number. The Chaincode Version Information page is displayed.
4. Click the Installed on Peers pane to see which peers the chaincode is installed on. You can click the peer to view more information about it.
5. Click the Instantiated on Channels pane to see which channels the chaincode is instantiated on. You can click a channel to view more information about it.

   From this pane, you can also instantiate a specific version of the chaincode version. If the chaincode was instantiated on a channel, then you can view its endorsement policy.

   Note that you can instantiate different versions of a chaincode on different channels.
6. **(New in v19.1.3)** Click the Private Data Collections pane to view the private data collections that were added when the chaincode was instantiated.

Upgrade a Chaincode

If a developer modifies a chaincode's source, then you'll need to deploy it to a new version of the chaincode. If needed, you can revert back to an older version of a chaincode.

You can instantiate different versions of the same chaincode on different channels.

You must be an administrator to perform this task.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the table lists all of the chaincodes installed on the network.
2. Locate the chaincode that you want to upgrade, click **More Actions**, and select **Upgrade**. The **More Actions** button only displays for chaincodes that have been instantiated.

The Upgrade Chaincode Step 1 of 2: Select a version page is displayed.

3. Select a version source. Note the following information:
   - Click **Select from existing versions** if you want to upgrade to a version that is already on the network. You might choose this option because the most current chaincode version contains errors and you need to temporarily use an older version until the chaincode can be fixed. Because the older version is on your system, the chaincode is already installed on the peers.
   - Choose **Install a new version** to upload the chaincode file. In the **Version** field enter a version number and in the **Target Peers** field, select the peers to install the chaincode on. In the **Chaincode Source** field, click **Upload Chaincode File** and browse for the chaincode ZIP file to upload.
4. Click **Next**.

   The Upgrade Chaincode Step 2 of 2: Upgrade page is displayed.
5. Decide if you want to instantiate the chaincode version now or later.
   - Click **Close** to close the wizard and upgrade later.
   - To upgrade now, select the channel to upgrade the chaincode on and the peers to instantiate the chaincode to. If required, enter initialize parameters, an endorsement policy, and transient map. See Specify an Endorsement Policy. Click **Next**.

   The chaincode is upgraded.

What Are Private Data Collections?

(19.1.3 and later versions only) Private data collections specify subsets of organizations that endorse, commit, or query private data on the channel.

Use private data collections in cases where you want a group of organizations on the channel to share data and to prevent the other organizations on the channel from seeing the data. Private data is distributed peer to peer and not by blocks, so the transaction data is kept confidential from the ordering service. Collections help you reduce the number of channels and their required maintenance on your network.

The primary components in a private data collection are:

- The private data that you specify in your private data collection definition. Private data is sent with the gossip protocol from peer to peer within the organizations that you specify in your policy. Private data is stored in a private database on the peer. The ordering service isn't used and can't see the private data.
- A hash of the data, which is endorsed, ordered, and written to each peer on the channel. This hash is evidence of the transaction and can be used for audit purposes.

When you instantiate a chaincode, you can associate it with one or more private data collections. Also when you instantiate a chaincode, you should specify a transient map to pass the private data from the client to the peers for endorsement. The collection definition specifies who can persist data, how many peers the data is distributed to, how many peers are required to disseminate the private data, and how long the private data is persisted in the private database.

Add Private Data Collections

(19.1.3 and later versions only) You can add private data collections to channels. Private data collections specify subsets of organizations that endorse, commit, or query private data on the channel.

Use private data collections in cases where you want a group of organizations on the channel to share data within a transaction and to prevent the other organizations on the channel from seeing the data.

If you're going to use private data collections across the organizations in your network, then you need to configure anchor peers. Anchor peers facilitate private data gossip among the organizations. See Add an Anchor Peer.

You specify the private data collections when you instantiate the chaincode.

1. Go to the console and select the Chaincodes tab.
2. Locate the chaincode that you want to instantiate and begin the instantiation process.
3. Expand the Private Data Collections section and add the collection definition as needed.

Field	Description
Collection Name	Enter the collection's name. You'll reference this name in the chaincode.
Policy	Create the policy to specify which organizations are included in the collection and which peers can store the private data. Each member listed in the policy must be included in an `OR` signature policy list. To support read/write transactions, the private data distribution policy must contain more organizations than the chaincode endorsement policy because peers must have the private data to endorse transactions. For example, in a channel with ten organizations, five of the organizations are included in a private data collection policy, but the endorsement policy requires three organizations to endorse a transaction.

Field	Description
Peers Required	Enter the number of peers that each endorsing peer must distribute private data to before the peer signs the endorsement and returns the proposal response. Oracle recommends that you set this value to 1 or more peers to: • Ensure redundancy of the private data on multiple peers in the network. • Ensure that private data is available if the endorsing peers become unavailable. Note that setting this value to 0 means that distribution isn't required. However, if the **Max Peer Count** field is set to greater than 0, then private data distribution might still occur.
Max Peer Count	Enter the maximum number of peers that the current endorsing peer attempts to distribute the data to. This is to ensure redundancy so that peers are available between endorsement time and commit time to pull the private data if an endorsing peer isn't available. If you set this value to 0, then the private data isn't distributes at the time of endorsement. This causes private data pulls against the endorsing peers on all authorized peers at commit time.
Block to Live	Enter the length in number of blocks that you want data to reside on the private database. The data is purged when the number of blocks is reached. Set this value to 0 if you never want to purge the data. Note that a peer can fail to pull private data from another peer if a private data collection's blocktolive value is less than 10, and its requiredPeerCount and maxPeerCount are less than the total number of peers in the channel. This is a known Hyperledger Fabric issue. See https://jira.hyperledger.org/browse/FAB-11889.

4. Click **Add New Collection** and your collection's information is displayed in the private data collection table.
5. If needed, specify other collections.
6. Complete the other fields on the Instantiate Chaincode page as needed.
7. Click **Instantiate**.

View Private Data Collections

(19.1.3 and later versions only) You can view information about a chaincode's private data collections.

After you instantiate a chaincode, you might need to view its private data collections to see how they were defined.

You can’t modify the private data collections for an instantiated chaincode. If you need to change the private data collections, then you need to upgrade the chaincode and specify new private data collections.

1. Go to the console and select the Chaincodes tab.

   The Chaincodes tab is displayed and the table lists the chaincodes installed on the network.
2. Locate the chaincode that you want to view private data collections for and expand it in the table.
3. Click the chaincode version that you want.

   The Chaincode Version Information page is displayed.
4. In the Private Data Collections tab, locate the collection that you want to view.

8 Develop Blockchain Applications

Blockchains require smart contracts (chaincode) to update the ledger. In addition, you will also require a client application that utilizes either the Oracle Blockchain Platform REST API or native Hyperledger Fabric SDK to interact with the blockchain directly. There are other operational and administrative tasks to consider, namely the creation of peers and channels and installation of chaincode.

Topics

- Before You Develop an Application
- Use the Hyperledger Fabric SDKs to Develop Applications
- Use the REST APIs to Develop Applications

Before You Develop an Application

Before you write an application, download and use the sample applications, and ensure that you've the correct certificates and privileges to run an application.

Oracle Blockchain Platform provides downloadable samples that help you understand how to write chaincodes and applications. See:

- What Are Chaincode Samples?
- Explore Oracle Blockchain Platform Using Samples

Oracle Blockchain Platform uses Hyperledger Fabric as its foundation. Use the Hyperledger Fabric documentation to help you write applications. The Key Concepts and Tutorials sections should be read before you write you own application: Welcome to Hyperledger Fabric.

Prerequisites for Application Development

A user ID and password for the application user must exist in your IDCS server. Depending on the functions in the application, this user must have the following:

- To install and instantiate chaincode:
  - You must have administrative access in order to install or deploy chaincode.
  - You must export the admincerts, cacerts, and tlscacerts certificates as described in Export Certificates so that they can be placed in your application in the peer and orderer nodes crypto folders.
  - You must export the admin credentials similarly to how you exported the certificates (from the action menu, select **Export Admin Credential**). This will download a ZIP file containing the signed certificate and keystore files that need to be placed in your application in the peer and orderer nodes crypto folders.
- To run operations against an installed and instantiated chaincode:

  - You must export the admincerts, cacerts, and tlscacerts certificates as described in Export Certificates so that they can be placed in your application in the peer node crypto folders.
  - You must export the tlscacerts certificate for the orderer node as described in Join the Participant or Scaled-Out OSNs to the Founder's Ordering Service so that it can be placed in your application.
  - The chaincode you're invoking must be installed and deployed to a channel and node that your user ID has access to.
  - A REST proxy node must be configured and the chaincode enabled for REST proxy access. The user ID and password for the node must be provided.
- To run functions against a REST API endpoint:
  - The chaincode you're invoking must be installed and deployed to a channel and node that your user ID has access to.
  - A REST proxy node must be configured and the chaincode enabled for REST proxy access. The user ID and password for the node must be provided.

Use the Hyperledger Fabric SDKs to Develop Applications

Applications use a software development kit (SDK) to access the APIs that permit queries and updates to the ledger. You can install and use the Hyperledger Fabric SDKs to develop applications for Oracle Blockchain Platform.

The REST APIs provided by Oracle Blockchain Platform have been created with maximum flexibility in mind; you can invoke a transaction, invoke a query, or view the status of a transaction. See REST API for Oracle Blockchain Platform.

However this means that you'll likely want to wrap the existing API endpoints in an application to provide object-level control. Applications can contain much more fine-grained operations.

SDK Versions

- If your Oracle Blockchain Platform founder instances were created using 19.2.3 or later, they support V1.4 of the Hyperledger Fabric SDKs.

Installing the Hyperledger Fabric SDK for Node.js

Information about how to use the Fabric SDK for Node.js can be found here: Hyperledger Fabric SDK for Node.js documentation

On the **Developer Tools** tab, open the **Application Development** pane.

- You can install the Hyperledger Fabric Node.js SDK from this tab.
- If you've previously installed it you must modify it to work with Oracle Blockchain Platform following the instructions in Update the Hyperledger Fabric SDKs to Work with Oracle Blockchain Platform.

Installing the Hyperledger Fabric SDK for Java

Information about how to use the Fabric SDK for Java can be found here: Hyperledger Fabric SDK for Java documentation

On the **Developer Tools** tab, open the **Application Development** pane.

- You can install the Hyperledger Fabric Java SDK from this tab.

- If you've previously installed it you must modify it to work with Oracle Blockchain Platform following the instructions in Update the Hyperledger Fabric SDKs to Work with Oracle Blockchain Platform.

Install a build tool such as Apache Maven.

Structuring your Application

Your Java application should be structured similar to the following:

```
/Application
  /artifacts
    /cypto
      /orderer
        Contains the certificates required for the application to act 
on the orderer node
        In participant instances only contains TLS certificates
      /peer
        Contains the certificates required for the application to act 
on the peer node
    /src
      chaincode.go if installing and deploying chaincode to the 
blockchain
  /java
    pom.xml or other build configuration files
    /resources
      Any resources used by the Java code, including artifacts such as 
the endorsement policy yaml file and blockchain configuration properties
    /src
      Java source files
```

Your Node.js application should be structured similar to the following:

```
/Application
  /artifacts
    /cypto
      /orderer
        Contains the certificates required for the application to act 
on the orderer node
        In participant instances only contains TLS certificates
      /peer
        Contains the certificates required for the application to act 
on the peer node
    /src
      chaincode.go if installing and deploying chaincode to the 
blockchain
  /node
    package.json file
    application.js
    /app
      Any javascript files called by the application
      /tools
```

Running the application

You're now ready to run and test the application. In addition to any status messages returned by your application, you can check the ledger in the Oracle Blockchain Platform console to see your changes:

1. Go to the Channels tab in the console and locate and click the name of the channel running the blockchain.
2. In the channel's **Ledger** pane, view the chaincode's ledger summary.

Update the Hyperledger Fabric SDKs to Work with Oracle Blockchain Platform

There's an incompatibility between an OCI infrastructure component and the Node.js and Java SDKs provided with Fabric. Follow the steps in this topic to correct this problem.

Methods of updating the Fabric SDKs

There are two ways of updating the SDK:

- Using Oracle scripts to download and install the Node.js SDK or Java SDK which will patch the code as it installs.
- Manually as described in this topic.

To use the scripts, on the console's **Developer Tools** tab, select the **Application Development** pane. The links to download both the Node.js SDK and Java SDK have updates built in which will patch the code as it installs.

- Fabric Java SDK: We've created an updated `grpc-netty-1.23.0.jar` file, which is the module referenced by the Java SDK which requires modifications.
- Fabric Node.js SDK: We have created the `npm_bcs_client.sh` script to replace the standard Fabric `npm install` operations that users would perform to download and install the Node.js Fabric client package. The script runs the same npm command, but it also patched the needed component and rebuilds it.

Manually updating the Fabric Node.js SDK

Do the following to rebuild the `grpc-node` module to connect the peers and orderers with grpcs client (via TLS).

1. Install `fabric-client` without executing the grpc module's build script:

   ```
   npm install --ignore-scripts fabric-client
   ```

2. On Windows, you need to disable ALPN explicitly
   - Update `node_modules/grpc/binding.gyp` by changing:

     ```
     '_WIN32_WINNT=0x0600'
     ```

     to

     ```
     '_WIN32_WINNT=0x0600','TSI_OPENSSL_ALPN_SUPPORT=0'
     ```

   - Due to the issue outlined in https://github.com/nodejs/node/issues/4932, to build `grpc-node` on Windows, you must first remove

`<node_root_dir>/include/node/openssl/`. Run the following to find your `<node_root_dir>`:

```
node-gyp configure
```

3. Rebuild `grpc`

```
npm rebuild --unsafe-perm --build-from-source
```

You can now install any other modules you need and run the project.

Manually updating the Fabric Java SDK

For `fabric-sdk-java`, do the following steps to rebuild the `grpc-netty` package to connect the peers and orderers with grpcs client (via tls). `grpc-netty` is a sub-project of `grpc-java`.

1. Install project dependencies:

   ```
   mvn install
   ```

2. Download grpc-java source code:

   ```
   git clone https://github.com/grpc/grpc-java.git
   ```

3. Find the `grpc` version that your `fabric-sdk-java` uses, and checkout the code. Different versions of `fabric-sdk-java` may use different version of `grpc`. Check `pom.xml` to find out what `grpc` version your `fabric-sdk-java` uses. For example, `fabric-sdk-java 1.4.11` uses `grpc-java 1.23.0` as found in its `pom.xml`: https://github.com/hyperledger/fabric-sdk-java/blob/v1.4.11/pom.xml.

   In the `grpc-java` directory, checkout the version of `grpc` that `fabric-sdk-java` uses:

   ```
   git checkout -b v1.23.0
   ```

4. Change the code to avoid an `alpn` error from the server side.
   - Change the target code of `grpc-java_root/netty/src/main/java/io/grpc/netty/ProtocolNegotiators.java`
   - In the function `userEventTriggered0` change:

     ```
     if
     (NEXT_PROTOCOL_VERSIONS.contains(handler.applicationProtocol()))
     {
     ```

     to

     ```
     if (handler.applicationProtocol() == null ||
     NEXT_PROTOCOL_VERSIONS.contains(handler.applicationProtocol())) {
     ```

The code will look similar to:

```
    @Override
    protected void userEventTriggered0(ChannelHandlerContext 
ctx, Object evt) throws Exception {
      ...
          if (handler.applicationProtocol() == null || 
NEXT_PROTOCOL_VERSIONS.contains(handler.applicationProtocol())) {
            // Successfully negotiated the protocol.
            logSslEngineDetails(Level.FINER, ctx, "TLS 
negotiation succeeded.", null);
      ...
    }
```

5. Build the project to generate the target patched package. Use gradle to build the `grpc-java` project. Or you can just rebuild the `grpc-netty` sub-project in the grpc netty directory `gradle build`.

   After the build is done, you can get the target patched jar package in the directory `grpc-java\netty\build\libs\grpc-netty-1.23.0.jar`.
6. Add the patched package into your Maven local repository.

   Replace official `grpc-netty` jar package with the patched package in either of the following two ways:
   - Use Maven to install the package by local file:

     ```
     mvn install:install-
     file -Dfile=local_patched_grpc_netty_package_root/grpc-
     netty-1.23.0.jar -DgroupId=io.grpc -DartifactId=grpc-netty -
     Dversion=1.23.0 -Dpackaging=jar
     ```

     You must keep the target `groupid`, `artifactid`, and `version` the same as the package you want to replace.
   - Manually replace your package. Go to the local Maven repository, find the directory where the target package is located, and replace the package with patched package.
7. Run the project.

Use the REST APIs to Develop Applications

The REST APIs provided by Oracle Blockchain Platform have been created with maximum flexibility in mind; you can invoke a transaction, invoke a query, or view the status of a transaction. However this means that you'll likely want to wrap the existing API endpoints in an application to provide object-level control. Applications can contain much more fine-grained operations.

Any application using the REST APIs requires the following:

- The chaincode name and version.
- The REST server URL and port, and the user ID and password for the REST node.
- Functions to invoke transactions against or query the ledger.

See REST API for Oracle Blockchain Platform for information on the existing operations, including examples and usage syntax.

Structuring your Application

Your REST API application should be structured similar to the following:

```
/Application
  /artifacts
    /crypto
      /orderer
        Contains the certificates required for the application to act
on the orderer node
        In participant instances only contains TLS certificates
      /peer
        Contains the certificates required for the application to act
on the peer node
    /src
  /REST
    Application script containing REST API calls
```

9 Work With Databases

This topic contains information to help you understand how to query the state database and how to create and configure a rich history database.

Topics:

- Query the State Database
- Create the Rich History Database

Query the State Database

This topic contains information to help you understand how to query the state database.

Topics:

- What's the State Database?
- Supported Rich Query Syntax
- State Database Indexes
- Differences in the Validation of Rich Queries

What's the State Database?

The blockchain ledger’s current state data is stored in the state database.

When you develop Oracle Blockchain Platform chaincodes, you can extract data from the state database by executing rich queries. Oracle Blockchain Platform supports rich queries by using the SQL rich query syntax and the CouchDB find expressions. See SQL Rich Query Syntax and CouchDB Rich Query Syntax.

Hyperledger Fabric doesn’t support SQL rich queries. If your Oracle Blockchain Platform network contains Hyperledger Fabric participants, then you need to make sure to do the following:

- If your chaincodes contain SQL rich query syntax, then those chaincodes are installed only on member peers using Oracle Blockchain Platform.
- If a chaincode needs to be installed on Oracle Blockchain Platform and Hyperledger Fabric peers, then use CouchDB syntax in the chaincodes and confirm that the Hyperledger Fabric peers are set up to use CouchDB as their state database repository. Oracle Blockchain Platform can process CouchDB.

How Does Oracle Blockchain Platform Work with Berkeley DB?

Oracle Blockchain Platform uses Oracle Berkeley DB as the state database. Oracle Blockchain Platform creates relational tables in Berkeley DB based on the SQLite extension. This architecture provides a robust and performant way to validate SQL rich queries.

For each channel chaincode, Oracle Blockchain Platform creates a Berkeley DB table. This table stores state information data, and contains at least a key column named `key`, and a value column named `value` or `valueJson`, depending on whether you're using JSON format data.

Column Name	Type	Description
`key`	TEXT	Key column of the state table.
`value`	TEXT	Value column of the state table.
`valueJson`	TEXT	JSON format value column of the state table.

Note that the `valueJson` and `value` columns are mutually-exclusive. So, if the chaincode assigns a JSON value to a key, then the `valueJson` column will hold that value, and the value column will be set to null. If the chaincode assigns a non-JSON value to a key, then the `valueJson` column will be set to null, and the value column will hold the value.

Example of a State Database

These are examples of keys and their values from the Car Dealer sample's state database:

key	value	valueJson
abg1234	null	{"docType": "vehiclePart", "serialNumber": "abg1234", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "airbag 2020", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}
abg1235	null	{"docType": "vehiclePart", "serialNumber": "abg1235", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "airbag 4050", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}
ser1236	null	{"docType": "vehiclePart", "serialNumber": "ser1236", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "seatbelt 10020", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}
bra1238	null	{"docType": "vehiclePart", "serialNumber": "bra1238", "assembler": "bobs-bits", "assemblyDate": 1502688979, "name": "brakepad 4200", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}
dtrt10001	null	{"docType": "vehicle", "chassisNumber": "dtrt10001", "manufacturer": "Detroit Auto", "model": "a coupe", "assemblyDate": 1502688979, "airbagSerialNumber": "abg1235", "owner": "Sam Dealer", "recall": false, "recallDate": 1502688979

State Database Indexes

The state database can contain a large amount of data. In such cases Oracle Blockchain Platform uses indexes to improve data access.

Default Indexes

When a chaincode is deployed, Oracle Blockchain Platform creates two indexes.

- Key index — Created on the key column.
- Value index — Created on the value column.

Custom Indexes

In some cases, you might need to create custom indexes. You define these indexes using any expression that can be resolved in the context of the state table. Custom indexes created against Berkeley DB rely on the SQLite syntax, but they otherwise follow the same CouchDB implementation provided by Hyperledger Fabric.

Note that you can use custom indexes to dramatically improve the performance of WHERE and ORDER BY statements on large data sets. Because using custom indexes slows down data insertions, you should use them judiciously.

Each custom index is defined as an array of expressions, which support compound indexes, expressed as a JSON document inside one file (note that there's one index per file). You must package this file with the chaincode in a folder named "indexes" in the following directory structure: `statedb/relationaldb/indexes`. See How to add CouchDB indexes during chaincode installation.

Example Custom Indexes

The custom index examples in this section use the Car Dealer sample.

Example 1 —This example indexes the use of the `json_extract` expression in the context of WHERE and ORDER BY expressions.

```
{"indexExpressions": ["json_extract(valueJson, '$.owner')"]}
```

For example:

```
SELECT … FROM … ORDER BY json_extract(valueJson, '$.owner')
```

Example 2 — This example indexes the compound use of the two `json_extract` expressions in the context of WHERE and ORDER BY expressions.

```
{"indexExpressions": ["json_extract(valueJson, '$.docType')",
"json_extract(valueJson, '$.owner')"]}
```

For example:

```
SELECT … FROM … WHERE json_extract(valueJson, '$.docType') = 'vehiclePart'
AND json_extract(valueJson, '$.owner') = 'Detroit Auto'
```

Example 3 — This example creates two indexes: the index described in Example 1 and the index described in Example 2. Note that each JSON structure needs to be included in a separate file. Each file describes a single index: a simple index like Example 1, or a compound index like Example 2.

Index 1: `{"indexExpressions": ["json_extract(valueJson, '$.owner')"]}`

Index 2: `{"indexExpressions": ["json_extract(valueJson, '$owner')", "json_extract(valueJson, '$.docType')"]}`

In the following example, Index 2 is applied to the `AND` expression in the `WHERE` portion of the query, while Index 1 is applied to the `ORDER BY` expression:

```
SELECT … FROM … WHERE json_extract(valueJson, '$.docType') = 'vehiclePart'
AND json_extract(valueJson, '$.owner') = 'Detroit Auto' ORDER BY
json_extract(valueJson, '$.owner')
```

JSON Document Format

The JSON document must be in the following format:

```
{"indexExpressions": [expr1, ..., exprN]}
```

For example:

```
{"indexExpressions": ["json_extract(valueJson, '$.owner')"]}
```

Differences in the Validation of Rich Queries

In some cases, the standard Hyperledger Fabric with CouchDB rich query and the Oracle Berkeley DB rich query behave differently.

In standard Hyperledger Fabric with CouchDB, each key and value pair returned by the query is added to the transaction's read-set and is validated at validation time and without re-executing the query. In Berkeley DB, the returned key and value pair isn’t added to the read-set, but the rich query's result is hashed in a Merkle tree and validated against the re-execution of the query at validation time.

Native Hyperledger Fabric doesn’t provide data protection for rich query. However, Berkeley DB contains functionality that protects and validates the rich query by adding the Merkle tree hash value into the read-set, re-executing the rich query, and at the validation stage re-calculating the Merkle tree value. Note that because validation is more accurate in Oracle Blockchain Platform with Berkeley DB, chaincode invocations are sometimes flagged for more frequent phantom reads.

Supported Rich Query Syntax

Oracle Blockchain Platform supports two types of rich query syntax that you can use to query the state database: SQL rich query and CouchDB rich query.

Topics:

- SQL Rich Query Syntax
- CouchDB Rich Query Syntax

SQL Rich Query Syntax

The Berkeley DB JSON extensions are in the form of SQL functions.

Before You Begin

Note the following information:

- You can only access the channel chaincode (<STATE>) that you’re executing your query from.
- Only the SELECT statement is supported.
- You can’t modify the state database table.
- A rich query expression can have only one SELECT statement.

- The examples in this topic are just a few ways that you can write your rich query. You've access to the usual full SQL syntax to query a SQL database.
- You've access to the JSON1 Extension (SQLite extension). See JSON1 Extension and SQL As Understood by SQLite.

If you need more information about writing and testing chaincodes, see Develop Chaincodes.

How to Refer to the State Database in Queries

The state database table name is internally managed by Oracle Blockchain Platform, so you don't need to know the state database's physical name when you write a chaincode.

Instead, you must use the `<STATE>` alias to refer to the table name. For example: `select key, value from <STATE>`.

Note that the `<STATE>` alias is **not** case-sensitive, so you can use either `<state>`, `<STATE>`, or something like `<StAtE>`.

Retrieve All Keys

Use this syntax:

```
SELECT key FROM <STATE>
```

For example, if you use this syntax to query the Car Dealer sample, then you'll get the following list of keys:

key

abg1234

abg1235

ser1236

bra1238

dtrt10001

Retrieve All Keys and Values Ordered Alphabetically by Key

Use this syntax:

```
SELECT key AS serialNumber, valueJson AS details FROM  <state> ORDER BY
key
```

For example, if you use this syntax to query the Car Dealer sample, then you'll get the following results:

serialNumber	details
abg1234	{"docType": "vehiclePart", "serialNumber": "abg1234", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "airbag 2020", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}

serialNumber	details
abg1235	{"docType": "vehiclePart", "serialNumber": "abg1235", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "airbag 4050", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}
bra1238	{"docType": "vehiclePart", "serialNumber": "bra1238", "assembler": "bobs-bits", "assemblyDate": 1502688979, "name": "brakepad 4200", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}
dtrt10001	{"docType": "vehicle", "chassisNumber": "dtrt10001", "manufacturer": "Detroit Auto", "model": "a coupe", "assemblyDate": 1502688979, "airbagSerialNumber": "abg1235", "owner": "Sam Dealer", "recall": false, "recallDate": 1502688979
ser1236	{"docType": "vehiclePart", "serialNumber": "ser1236", "assembler": "panama-parts", "assemblyDate": 1502688979, "name": "seatbelt 10020", "owner": "Detroit Auto", "recall": false, "recallDate": 1502688979}

Retrieve All Keys and Values Starting with “abg”

Use this syntax:

```
SELECT key AS serialNumber, valueJson AS details FROM <state> WHERE key
LIKE 'abg%'SELECT key, value FROM <STATE>
```

For example, if you use this syntax to query the Car Dealer sample, then you’ll get the following results:

serialNumber	details
abg1234	{"docType": "vehiclePart", "serialNumber": "abg1234", "assembler": "panama-parts", "assemblyDate": "1502688979", "name": "airbag 2020", "owner": "Detroit Auto", "recall": "false", "recallDate": "1502688979"}
abg1235	{"docType": "vehiclePart", "serialNumber": "abg1235", "assembler": "panama-parts", "assemblyDate": "1502688979", "name": "airbag 4050", "owner": "Detroit Auto", "recall": "false", "recallDate": "1502688979"}

Retrieve All Keys with Values Containing a Vehicle Part Owned by "Detroit Auto"

Use this syntax:

```
SELECT key FROM <state> WHERE json_extract(valueJson, '$.docType') =
'vehiclePart' AND json_extract(valueJson, '$.owner') = 'Detroit Auto'
```

For example, if you use this syntax to query the Car Dealer sample, then you’ll get the following list of keys:

key
abg1234
abg1235
ser1236
bra1238

Retrieve Model and Manufacturer for all Cars Owned by "Sam Dealer"

Use this syntax:

```
SELECT json_extract(valueJson, '$.model') AS model,
json_extract(valueJson, '$.manufacturer') AS manufacturer FROM
<state> WHERE json_extract(valueJson, '$.docType') = 'vehicle' AND
json_extract(valueJson, '$.owner') = 'Sam Dealer'
```

For example, if you use this syntax to query the Car Dealer sample, then you'll get the following results:

model	manufacturer
a coupe	Detroit Auto

If the state value is JSON array, you may use this syntax to retrieve model and manufacturer for all cars owned by "Sam Dealer":

```
SELECT json_extract(j.value, '$.model') AS model, json_extract(j.value,
'$.manufacturer') AS manufacturer FROM <state> s,
json_each(json_extract(s.valueJson,'$')) j WHERE json_valid(j.value) AND
json_extract(j.value, '$.owner') = 'Sam Dealer'
```

CouchDB Rich Query Syntax

Use the information in this topic if you're migrating your chaincodes containing CouchDB syntax to Oracle Blockchain Platform, or if you need to write chaincodes to install on Hyperledger Fabric peers participating in an Oracle Blockchain Platform network.

If you're writing a new chaincode, then Oracle recommends that you use SQL rich queries to take advantage of the performance benefits that Oracle Blockchain Platform with Berkeley DB provides.

If you need more information about writing and testing chaincodes, see Develop Chaincodes.

Unsupported Query Parameters and Selector Syntax

Oracle Blockchain Platform doesn't support the `use_index` parameter. If used, Oracle Blockchain Platform ignores this parameter, and it will automatically pick the indexes defined on the StateDB in question.

Parameter	Type	Description
use_index	json	Instructs a query to use a specific index.

Retrieve All Models, Manufacturers, and Owners of Cars, and Order Them by Owner

Use this expression:

```
{
  "fields": ["model", "manufacturer", "owner"],
  "sort": [
    "owner"
```

```
   ]
}
```

Retrieve Model and Manufacturer for All Cars Owned by “Sam Dealer”

Use this expression:

```
{
  "fields": ["model", "manufacturer"],
  "selector": {
    "docType"  : "vehicle",
     "owner" : "Sam Dealer"
  }
}
```

Create the Rich History Database

This topic contains information to help you specify an Oracle database connection and choose channels to create the rich history database. You’ll use this database to make analytics reports and visualizations of your ledger’s activities.

Topics:

What's the Rich History Database?

The rich history database is external to Oracle Blockchain Platform and contains data about the blockchain ledger’s transactions on the channels you select. You use this database to create analytics reports and visualization about your ledger’s activities.

For example, using the rich history database, you could create analytics to learn the average balance of all of the customers in your bank over some time interval, or how long it took to ship merchandise from a wholesaler to a retailer.

Internally, Oracle Blockchain Platform uses the Hyperledger Fabric history database to manage the ledger and present ledger transaction information to you in the console. Only the chaincodes can access this history database, and you can’t expose the Hyperledger Fabric history database as a data source for analytical queries. The rich history database uses an external Oracle database and contains many details about every transaction committed on a channel. This level of data collection makes the rich history database an excellent data source for analytics. For information about the data that the rich history database collects, see Rich History Database Tables and Columns.

You can only use an Oracle database such as Oracle Autonomous Data Warehouse or Oracle Database Classic Cloud Service with Oracle Cloud Infrastructure to create your rich history database. You use the Oracle Blockchain Platform console to provide the connection string and credentials to access and write to the Oracle database. Note that the credentials you provide are the database's credentials and Oracle Blockchain Platform doesn't manage them. After you create the connection, you'll select the channels that contain the ledger data that you want to include in the rich history database. See Enable and Configure the Rich History Database.

You can use standard tables or blockchain tables to store the rich history database. Blockchain tables are tamperproof append-only tables, which can be used as a secure ledger while also being available for transactions and queries with other tables. For more information, see Oracle Blockchain Table.

You can use any analytics tool, such as Oracle Analytics Cloud or Oracle Data Visualization Cloud Service, to access the rich history database and create analytics reports or data visualizations.

Create the Oracle Database Classic Cloud Service Connection String

You must collect information from the Oracle Database Classic Cloud Service deployed on Oracle Cloud Infrastructure to build the connection string required by the rich history database. You must also enable access to the database through port 1521.

Find and Record Oracle Database Classic Cloud Service Information

The information you need to create a connection to the Oracle Database Classic Cloud Service is available in the Oracle Cloud Infrastructure Console.

1. From the Infrastructure Console, click the navigation menu in the top left corner, and then click **Database**.
2. Locate the database that you want to connect to and record the **Public IP** address.
3. Click the name of the database that you want to connect to and record the values in these fields:
    - **Database Unique Name**
    - **Host Domain Name**
    - **Port**
4. Find a username and password of a database user with permissions to read from this database, and make a note of these. For example, the user SYSTEM.

Enable Database Access Through Port 1521

Add an ingress rule that enables the rich history database to access the database through port 1521.

1. In the Oracle Cloud Infrastructure home page, click the navigation icon and then under **Databases** click **DB Systems**.
2. Click the database that you want to connect to.
3. Click the **Virtual Cloud Network** link.

4. Navigate to the appropriate subnet, and under **Security Lists**, click **Default Security List For <Target Database>**.

   The Security List page is displayed.
5. Click **Edit All Rules**.
6. Add an ingress rule to allow any incoming traffic from the public internet to reach port 1521 on this database node, with the following settings:
   - **SOURCE CIDR**: 0.0.0.0/0
   - **IP PROTOCOL**: TCP
   - **SOURCE PORT RANGE**: All
   - **DESTINATION PORT RANGE**: 1521
   - **Allows**: TCP traffic for ports: 1521

Build the Connection String

After enabling access to the Oracle database, use the information you collected to build the connection string in the Configure Rich History dialog.

Construct the connection string like this: *<publicIP>*:*<portNumber>*/*<database unique name>.<host domain name>*

For example, 123.213.85.123:1521/
CustDB_iad1vm.sub05031027070.customervcnwith.oraclevcn.com

Ensure the Database User has Correct Privileges

In order for the rich history functionality to be able to manage its database sessions and to recover from temporary database or network downtime, ensure the database user registered with Oracle Blockchain Platform has the following two privileges:

```
grant select on v_$session to <user>;
grant alter system to <user>;
```

If the database user doesn't have those already, they will need to be granted by the system database administrator.

Without these privileges Oracle Blockchain Platform will be able to replicate to the database but it won't be able to recover from situations leading to a damaged database session, preventing the rich history from catching up with recent transactions for an extended period.

Enable and Configure the Rich History Database

Use the console to provide database connection information and select the channels with the chaincode ledger data that you want to write to the rich history database. By default channels aren’t enabled to write data to the rich history database.

Note the following information:

- Each blockchain network member configures its own rich history database.
- You must use an Oracle database. No other database types are supported.
- Each channel that writes to the rich history database must contain at least one peer node.

1. Enter connection and credential information for the Oracle database that you want to use to store rich history information.
   a. Go to the console and click the **Options** button and click **Configure Rich History**. This button is located above the bar that contains the tabs that you use to navigate to nodes, channels, and chaincodes.

      The Configure Rich History dialog box is displayed.
   b. Enter the user name and password required to access the Oracle database.
   c. In the **Connection String** field, enter the connection string for the database that you'll use to store rich history data. What you enter here depends on the Oracle database you're using.
      - If you're using Oracle Autonomous Data Warehouse, then you'll enter something similar to <*username*>adw_high. To find Oracle Autonomous Data Warehouse's connection information, go to its credential wallet ZIP file and open its TNS file.
      - If you're using Oracle Database Classic Cloud Service with Oracle Cloud Infrastructure, see Create the Oracle Database Classic Cloud Service Connection String.
      - If you're using a non-autonomous Oracle database (a database that doesn't use a credential wallet) and want to use the `sys` user to connect to the database, then you must append `?as=sys[dba|asm|oper]` to the connection string. For example, `123.123.123.123:1521/example.oraclevcn.com?as=sysdba`
   d. If you're using an Oracle Cloud autonomous database instance (for example, Oracle Autonomous Data Warehouse or Oracle Autonomous Transaction Processing), then use the **Wallet Package File** field to upload the required credential wallet ZIP file. This file contains client credentials and is generated from the Oracle autonomous database.

      Note:

      When you open the Configure Rich History dialog box again after you configure rich history, the wallet file name is not displayed. If you update other settings, you must upload the wallet ZIP file again before clicking **Save**. If you click **Save** while no wallet file name is displayed, the configuration is updated not to use a wallet file.
   e. To use blockchain tables to store the rich history database, select **Use Database Blockchain Table**.

      The underlying database must support blockchain tables. For more information, see Oracle Blockchain Table.
      - To specify the number of days to retain tables and rows, select **Basic Configuration**, and then enter the number of days to retain tables and rows. Enter `0` to retain tables or rows permanently. To prevent further changes to the retention values, select **Locked**.
      - To specify table and row retention by using a data definition language (DDL) statement, select **Advanced Configuration Query** and then enter the DDL statement.
   f. Click **Save**.

2. Enable rich history on the channels that contain the chaincode data that you want to write to the rich history database.
   a. Go to the console and select the **Channels** tab.
   b. Locate the channel that contains the chaincode data that you want to write to the rich history database. Click its **More Options** button and select **Configure Rich History**.

      The Configure Rich History dialog is displayed.
   c. Click the **Enable Rich History** checkbox. To add transaction details to the rich history database, select the details that you want added. Click **Save**.

The rich history database is configured, but tables are not created in the database immediately. When the next relevant transaction or ledger change happens, the tables are created in the rich history database.

Modify the Connection to the Rich History Database

You can change the rich history database's connection information.

After tables are created in the database for a channel, modifying the rich history configuration for the channel has no effect, even after you click **Save**, unless you change the user name and password or the connection string. If you change the user name and password, tables are created in the same database. If you change the connection string and credentials, a different database is configured, and tables are created after the next relevant transaction or ledger change. You cannot change a rich history database from standard tables to blockchain tables, and you cannot change retention times, unless you also change the credentials or connection string.

1. Go to the console and click the **Options** button and click **Configure Rich History**. This button is located above the bar that contains the tabs that you use to navigate to nodes, channels, and chaincodes.
2. If needed, update the user name and password required to access the Oracle database.
3. If needed, in the **Connection String** field, modify the connection string for the database that you'll use to store rich history data. What you enter here depends on the Oracle database you're using.
   - If you're using Oracle Autonomous Data Warehouse, then you'll enter something similar to <*username*>adw_high. To find Oracle Autonomous Data Warehouse's connection information, go to its credential wallet ZIP file and open its TNS file.
   - If you're using Oracle Database Classic Cloud Service with Oracle Cloud Infrastructure, see Create the Oracle Database Classic Cloud Service Connection String.
   - If you're using a non-autonomous Oracle database (a database that doesn't use a credential wallet) and want to use the `sys` user to connect to the database, then you must append `?as=sys[dba|asm|oper]` to the connection string. For example, `123.123.123.123:1521/example.oraclevcn.com?as=sysdba`
4. If you're using an Oracle Cloud autonomous database instance (for example, Oracle Autonomous Data Warehouse or Oracle Autonomous Transaction Processing), then use the **Wallet Package File** field to upload or re-upload the

required credential wallet file. This file contains client credentials and is generated from the Oracle autonomous database.

Note:

When you open the Configure Rich History dialog box again after you configure rich history, the wallet file name is not displayed. If you update other settings, you must upload the wallet ZIP file again before clicking **Save**. If you click **Save** while no wallet file name is displayed, the configuration is updated not to use a wallet file.

5. To use blockchain tables to store the rich history database, select **Use Database Blockchain Table**.

   The underlying database must support blockchain tables.

   - To specify the number of days to retain tables and rows, select **Basic Configuration**, and then enter the number of days to retain tables and rows. Enter `0` to retain tables or rows permanently. To prevent further changes to the retention values, select **Locked**.
   - To specify table and row retention by using a data definition language (DDL) statement, select **Advanced Configuration Query** and then enter the DDL statement.
6. Click **Save**.

Configure the Channels that Write Data to the Rich History Database

You can enable channels to write chaincode ledger data to the rich history database, and you can stop channels from writing data to the rich history database. You can also configure an individual channel to use a different rich history database configuration than the global setting.

You must specify the global information to connect to the rich history database before you can select channels that write to the rich history database. See Enable and Configure the Rich History Database.
After tables are created in the database for a channel, modifying the rich history configuration for the channel has no effect, even after you click **Save**, unless you change the user name and password or the connection string. If you change the user name and password, tables are created in the same database. If you change the connection string and credentials, a different database is configured, and tables are created after the next relevant transaction or ledger change. You cannot change a rich history database from standard tables to blockchain tables, and you cannot change retention times, unless you also change the credentials or connection string.

1. Go to the console and select the **Channels** tab.
2. Locate the channel that you want to modify access for. Click its **More Options** button and select **Configure Rich History**.

   The Configure Rich History dialog is displayed.
3. To enable collection of rich history data for the channel, select the **Enable Rich History** check box. To disable collection of rich history data for the channel, clear the **Enable Rich History** check box.

4. To configure the channel to collect rich history data using a different database or different settings, select **Use channel level configuration**, and then specify the settings to use.

   For more information about the rich history settings, see Enable and Configure the Rich History Database.

5. Click **Save**.

Monitor the Rich History Status

After configuring the rich history database, you can use the console to monitor the rich history replication status.

1. Go to the console and select the **Channels** tab.
2. In the channels table, click the **More Actions** button for the channel that you want to monitor, and then click **Rich History Status**.

   The Rich History Status dialog box is displayed, which includes details about replication and configuration status.

3. Click **Refresh** to display the latest status.

Limit Access to Rich History

You can use channel policies and access control lists (ACLs) to limit the organizations that can configure the rich history database and retrieve rich history status or configuration information.

By default, all organizations that have administrative access to a channel can configure rich history collection and can retrieve rich history status and configuration details. To limit this access to, for example, the founder organization, you create a channel policy and apply the policy to the resources that control access.

1. Go to the console and select the **Channels** tab.

   The **Channels** tab is displayed. The channel table contains a list of all of the channels on your network.

2. In the channel table, click the name of the channel where you want to limit access.
3. Click **Channel Policies**, and then create a signature policy that includes the organization members that will access the rich history functions.

   For more information about channel policies, see What Are Channel Policies?.

   For example, create a policy that includes only the identity of the founder organization, not the identity of any participant organizations.

4. Click **ACLs**.
5. In the Resources table, locate the resource that you want to update to use the new policy. Click **Expand** for the resource and then select the policy to assign to the resource

   The following table shows the resources that control access to rich history.

Resource	Access control
`obpadmin/ConfigureRichHistoryChannel`	Controls configuring, enabling, and disabling rich history for a channel.

Resource	Access control
`obpadmin/ GetRichHistoryChannelStatus`	Controls retrieving rich history replication status for a channel.
`obpadmin/ GetRichHistoryChannelConfig`	Controls retrieving the current rich history configuration for a channel.

6. Click **Update ACLs**.

The rich history access is now controlled by the new policy. Organization members that are not included in the new policy will receive an error message when they attempt to access a resource that is controlled by the policy.

Rich History Database Tables and Columns

The rich history database contains three tables for each channel: history, state, and latest height. You'll query the history and state tables when you create analytics about your chaincodes' ledger transactions. If you've chosen to select any of the transaction details when enabling the rich history, an additional table will be created with the transaction details.

History Table

The <*instanceName*><*channelName*>_hist table contains ledger history. The data in this table tells you the chaincode ID, key used, if the transaction was valid, the value assigned to the key, and so on.

Note that the **value** and **valueJson** columns are used in a mutually exclusive way. That is when a key value is valid json, then the value is set into the **valueJson** column. Otherwise the value is set in the **value** column. The **valueJson** column is set up as a json column in the database, which means users can query that column using the usual Oracle JSON specific extensions.

Column	Datatype
chaincodeId	VARCHAR2 (256)
key	VARCHAR2 (1024)
txnIsValid	NUMBER (1)
value	VARCHAR2 (4000)
valueJson	CLOB
blockNo	NUMBER NOT NULL
txnNo NUMBER	NOT NULL
txnId	VARCHAR2 (128)
txnTimestamp	TIMESTAMP
txnIsDelete	NUMBER (1)

State Table

The <*instanceName*><*channelName*>_state table contains data values replicated from the state database. You'll query the state table when you create analytics about the state of the ledger.

Note that the **value** and **valueJson** columns are used in a mutually exclusive way. That is when a key value is valid json, then the value is set into the **valueJson** column. Otherwise the value is set in the **value** column. The **valueJson** column is set up as a json column in the database, which means users can query that column using the usual Oracle JSON specific extensions.

Column	Datatype
chaincodeId	VARCHAR2 (256)
key	VARCHAR2 (1024)
value	VARCHAR2 (4000)
valueJson	CLOB
blockNo	NUMBER
txnNo	NUMBER

Latest Height Table

The <*instanceName*><*channelName*>_last table is used internally by Oracle Blockchain Platform to track the block height recorded in the rich history database. It determines how current the rich history database is and if all of the chaincode transactions were recorded in the rich history database. You can't query this database for analytics.

Transaction Details Table

The <*instanceName*><*channelName*>_more table contains attributes related to committed transactions. When enabling the rich history database, you can select which of these attributes you want to record in this table. The transaction details table only captures information about endorser transactions - not configuration transactions or any other kind of Hyperledger Fabric transactions.

Column	Datatype
CHAINCODEID	VARCHAR2 (256)
BLOCKNO	NUMBER
TXNNO	NUMBER
TXNID	VARCHAR2(128)
TXNTIMESTAMP	TIMESTAMP
SUBMITTERCN	VARCHAR2(512)
SUBMITTERORG	VARCHAR2(512)
SUBMITTEROU	VARCHAR2(512)
CHAINCODETYPE	VARCHAR2(32)
VALIDATIONCODENAME	VARCHAR2(32)
ENDORSEMENTS	CLOB
INPUTS	CLOB
EVENTS	CLOB
RESPONSESTATUS	NUMBER(0)
RESPONSEPAYLOAD	VARCHAR2(1024)
RWSET	CLOB
BLOCKCREATORCN	VARCHAR2(512)
BLOCKCREATORORG	VARCHAR2(512)
BLOCKCREATOROU	VARCHAR2(512)
CONFIGBLOCKNUMBER	NUMBER(0)
CONFIGBLOCKCREATORCN	VARCHAR2(512)

Column	Datatype
CONFIGBLOCKCREATORORG	VARCHAR2(512)
CONFIGBLOCKCREATOROU	VARCHAR2(512)

Note:

- Organization (ORG) and organization unit (OU) are driven by identity certificates, which implies that they may be assigned to multiple values. They are captured as a comma separated list in the table's values.
- For identities, the table includes information only about the "Subject" portion of the certificates, not the "Issuer" one.
- The `RWSET` column contains operations on all chaincodes (in the same ledger) performed during endorsement. As such, you will typically see both lscc read operations and the actual chaincode namespace operations.

A
Node Configuration

This topic contains information to help you understand and configure your nodes. Each node type has different configuration options.

Topics:

CA Node Attributes

A certificate authority (CA) node keeps track of identities and certificates on the blockchain network.

Only Administrators can change a node’s attributes. If you've got User privileges, then you can view a node’s attributes.

Table A-1 CA Node Attributes

Attribute	Description	Default Value
Fabric CA ID	This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it. You can’t modify this ID.	ca
Listen Port	This is the listening port that Oracle Blockchain Platform assigned to the node. You can’t change the port number.	Specific to your organization.
Max Enrollments	Use this field to determine how many times the CA server allows a password to be used for enrollment on the network. Consider the following options: • -1 — The server allows a password to be used an unlimited number of times for enrollment.	-1
Log Level	Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use INFO.	INFO

Console Node Attributes

The console node manages the performance of the console.

Only Administrators can change a node’s attributes. If you've got User privileges, then you can view a node’s attributes.

Table A-2 Console Node Attributes

Attribute	Description	Default Value
Console ID	This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it.	console
Local MSP ID	This is the assigned MSP ID for your organization. You can’t modify this ID.	NA
Listen Port	This is the listening port that Oracle Blockchain Platform assigned to the node. You can’t change the port number.	Specific to your organization.
Log Level	Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use ERROR.	INFO
Request Timeout (s)	Specify the maximum amount of time in seconds that you want the console to attempt to contact the nodes before timing out.	600

Orderer Node Attributes

An orderer node collects transactions from peer nodes, bundles them, and submits them to the blockchain ledger. The node’s attributes determine how the node performs and behaves on the network.

Only Administrators can change a node’s attributes. If you've got User privileges, then you can view a node’s attributes.

Table A-3 Orderer Node — General Attributes

Attribute	Description	Default Value
Orderer ID	This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it.	orderer<number-partition>
Local MSP ID	This is the assigned MSP ID for your organization. You can’t modify this ID.	NA

Table A-3 (Cont.) Orderer Node — General Attributes

Attribute	Description	Default Value
Listen Port	This is the listening port that Oracle Blockchain Platform assigned to the node. You can’t change the port number.	Specific to your organization.
Log Level	Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use ERROR.	INFO

Table A-4 Orderer Node — Advanced Attributes — Raft/Cluster tab

Attribute	Description	Default Value
SendBufferSize	The maximum number of messages in the egress buffer. Consensus messages are dropped if the buffer is full, and the transaction messages are waiting for space to be freed.	10
DialTimeout in seconds	The maximum duration of time after which connection attempts are considered as failed.	5
RPCTimeout in seconds	The maximum duration of time after which RPC attempts are considered as failed.	7
Replication/BufferSize in bytes	The maximum number of bytes that can be allocated for each in-memory buffer used for block replication from other cluster nodes.	20971520
Replication/ BackgroundRefreshInterval in minutes	The time between two consecutive attempts to replicate existing channels that this node was added to, or channels that this node failed to replicate in the past.	5
Replication/RetryTimeout in seconds	The maximum duration the ordering node will wait between two consecutive attempts.	5
Replication/PullTimeout in seconds	The maximum duration the ordering node will wait for a block to be received before it aborts.	5

Table A-4 (Cont.) Orderer Node — Advanced Attributes — Raft/Cluster tab

Attribute	Description	Default Value
Consensus/EvictionSuspicion in minutes	The threshold that a node will start suspecting its own eviction if it has been leaderless for this period of time.	10

Peer Node Attributes

A peer node reads, endorses, and writes transactions to the blockchain ledger. The node's attributes determine how the node performs and behaves on the network.

Only Administrators can change a node's attributes. If you've got User privileges, then you can view a node's attributes.

Table A-5 Peer Node — General Attributes

Attribute	Description	Default Value
Peer ID	This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it.	peer0
Local MSP ID	This is the assigned MSP ID for your organization. You can't modify this ID.	Specific to your organization.
Role	Specifies if the peer's role is Member or Admin. In most cases this field displays Member. This role is used by the chaincode's endorsement policy. The endorsement policy specifies the MSP that must validate the identity of the signer peer and the signer peer's role. The Admin role is normally assigned in situations where you want to further protect sensitive operations and make sure that those operations are endorsed by specific peers. The peers created with your instance were assigned the Member role.	Member
Listen Port	This is the listening port that Oracle Blockchain Platform assigned to the node. You can't change the port number.	Specific to your organization.

Table A-5 (Cont.) Peer Node — General Attributes

Attribute	Description	Default Value
Log Level	Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use ERROR.	INFO

Table A-6 Peer Node — Advanced Attributes — Gossip tab

Attribute	Description	Default Value
Bootstrap Peers	Provide the service name address and port that the peer uses to contact other peers during startup. This endpoint must match the endpoints of the peers in the same organization.	NA
Max Block Count to Store	Enter the maximum number of blocks to store in memory.	3
Max Propagation Burst Latency in milliseconds	Enter how many milliseconds between message pushes.	2
Max propagation burst size	Enter the number of messages to be stored until a push remote peer is triggered.	4
Propagate Iterations	Enter the number of times a message is pushed to the peers.	7
Propagate Peer Number	Enter how many peers to send messages to.	8
Pull Interval in seconds	Enter how many seconds between pull phases.	10
Pull Peer Number	Enter the number of peers to pull from.	9
Request State Info Interval in seconds	Enter how often to pull state information messages from the peers.	20
Publish state Info Interval in seconds	Enter how often to send state information messages to the peers.	21
Publish Cert Period in seconds	Enter how many seconds from startup that certificates are included in alive messages.	40
Dial Timeout in seconds	Enter how many seconds before dial times out.	10
Connect Timeout in seconds	Enter how many seconds until the connection times out.	20
Receive Buffer Size	Enter the size of the buffer for received messages.	20

Table A-6 (Cont.) Peer Node — Advanced Attributes — Gossip tab

Attribute	Description	Default Value
Send Buffer Size	Enter the size of the buffer for sending messages.	40
Digest Wait Time in seconds	Enter how many seconds to wait before the pull engine processes incoming digests.	15
Request Wait Time in seconds	Enter how many seconds to wait before the pull engine removes incoming nonce.	10
Response Wait Time in seconds	Enter how many seconds that the pull engine waits before it terminates the pull.	20
Alive Time Interval in seconds	Enter how often to check alive time.	15
Alive Expiration Timeout in seconds	Enter how many seconds to wait before the alive expiration times out.	12
Reconnect Interval in seconds	Enter how many seconds to wait before reconnecting.	9
Skip Block Verification	Click to skip block verification.	Selected

Table A-7 Peer Node — Advanced Attributes — Gossip/Election tab

Attribute	Description	Default Value
Membership Sample Interval in seconds	How often in seconds the peer checks its stability on the network.	3
Leader Alive Threshold in seconds	The number of seconds to elapse before the last declaration message is sent and before the peer determines leader election.	2
Leader Election Duration in seconds	The number of seconds to elapse after the peer sends the propose message and declares itself leader.	5

Table A-7 (Cont.) Peer Node — Advanced Attributes — Gossip/Election tab

Attribute	Description	Default Value
Leader	A channel's leader peer receives blocks and distributes them to the other peers within the cluster. Specify the mode that you want the peer to use to determine a leader. • **OrgLeader** — Select this option to use static leader mode and make the peer the organization leader. If you select this option and then add more peers to the channel, then you must set all peers to **OrgLeader**. • **UseLeaderElection** — Select this option to use dynamic leader election on the channel. Before an active leader is selected for the organization, the system must run the configuration transaction to add the organization to the channel, and then the system updates the new peers with the configuration transaction.	UseLeaderElection

Table A-8 Peer Node — Advanced Attributes — Event Service tab

Attribute	Description	Default Value
Buffer Size	Enter the maximum number of events that the buffer can contain. The system won't send the events that exceed this number.	100
Timeout in milliseconds	Enter in milliseconds the maximum time allowed for the business network to send an event.	1000

Table A-9 Peer Node — Advanced Attributes — Chaincode tab

Attribute	Description	Default Value
Startup timeout in seconds	Enter in seconds the maximum time to wait between when the container starts and the registry responds.	300

Table A-9 (Cont.) Peer Node — Advanced Attributes — Chaincode tab

Attribute	Description	Default Value
Execute timeout in seconds	Enter in seconds the maximum time that a chaincode attempts to execute before timing out.	30
Mode	Displays how the system runs the chaincode. This value is always net.	net
Keepalive in seconds	If you're using a proxy for communication, then enter in seconds the maximum amount of time to keep the connection between a peer and the chaincode alive.	0
Log Level	Specify the log level that you want to use for all loggers in the chaincode container. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use ERROR.	INFO
Shim Level	Specify the log level that you want to use for the shim logger.	WARNING

REST Proxy Node Attributes

A REST proxy node allows you to query or invoke a chaincode through the RESTful protocol. The node’s attributes determine how the node performs on the network and which channel, chaincode, and peers are used in the transactions performed by the node.

Only Administrators can change a node’s attributes. If you've got User privileges, then you can view a node’s attributes.

Table A-10 REST Proxy Node Attributes

Attribute	Description	Default Value
REST Proxy Name	This is the identifier or name that Oracle Blockchain Platform assigned the node when it created it. You can’t modify this ID.	restproxy
Proposal Wait Time (ms)	Enter the number of milliseconds that the node waits for completion of the proposal process. If this number is exceeded, then the transaction times out.	60,000

Table A-10 (Cont.) REST Proxy Node Attributes

Attribute	Description	Default Value
Transaction Wait Time (ms)	Enter the number of milliseconds that the node waits for execution after the transaction is submitted. If this number is exceeded, then the transaction times out.	300,000
Log Level	Specify the log level that you want to use for the node. Oracle suggests that for development or testing, you use DEBUG. And that for production, you use WARNING or ERROR.	INFO

B

Using the Fine-Grained Access Control Library Included in the Marbles Sample

Starting in v1.2, Hyperledger Fabric provided fine-grained access control to many of the management functions. Oracle Blockchain Platform now provides a updated version of the mables sample package on the Developer Tools tab of the console, implementing a library of functions that chaincode developers can use to create access control lists for chaincode functions. It currently only supports the Go language.

Topics

Background

The goal of this sample access control library is to provide the following:

- Provides a mechanism to allow you to control which users can access particular chaincode functions.
- The list of users and their entitlements should be dynamic and shared across chaincodes.
- Provides access control checks so that a chaincode can check the access control list easily.
- At chaincode deployment time, allows you to populate the list of resources and access control lists with your initial members.
- An access control list must be provided to authorize users to perform access control list operations.

Download the Sample

On the **Developer Tools** tab, open the **Samples** pane. Click the download link under **Marbles with Fine-Grained ACLs**. This package contains three sub-packages:

- `Fine-GrainedAccessControlLibrary.zip`:
  The fine-grained access control library. It contains functions in Go which can be used by chaincode developers to create access control lists for chaincode functions.
- `fgACL_MarbleSampleCC.zip`:
  The marbles sample with access control lists implemented. It includes a variety of functions to let you examine how to work with fine-grained access control lists, groups and resources to restrict functions to certain users/identities.
- `fgACL-NodeJSCode.zip`:

Node.js scripts which use the Node.js SDK to run the sample. `registerEnrollUser.js` can be used to register new users with the Blockchain Platform. `invokeQueryCC.js` can be used to run transactions against a Blockchain Platform instance.

Terminology and Acronyms

Term	Description
Identity	An X509 certificate representing the identity of either the caller or the specific identity the chaincode wants to check.
Identity Pattern	A pattern that matches one or more identities. The following patterns are suggested: • X.509 Subject Common Name – CN • X.509 Subject Organizational Unit – OU • X.509 Subject Organization – O • Group as defined in this library – GRP • Attribute – ATTR The format for a pattern is essentially just a string with a prefix. For example, to define a pattern that matches any identity in organization "example.com", the pattern would be "%O%example.com".
Resource	The name of anything the chaincode wants to control access to. To this library it is just a named arbitrary string contained in a flat namespace. The semantics of the name are completely up to the chaincode.
Group	A group of identity patterns.
ACL	Access Control List: a named entity that has a list of identity patterns, a list of types of access such as "READ", "CREATE", "INVOKE", "FORWARD", or anything the chaincode wants to use. This library will use access types of CREATE, READ, UPDATE, and DELETE (standard CRUD operations) to maintain its information. Other than those four as they relate to the items in this library, they are just strings with no implied semantics. An application may decide to use accesses of "A", "B", and "CUSTOM".

Fine-Grained Access Control Library Functions

The library package provides the following functions for Resources, Groups and ACLs as well as global functions.

Global Functions

Function	Description
Initialization(identity *x509.Certificate, stub shim.ChaincodeStubInterface) (error) (error)	When the chaincode is instantiated, the Initialization function is called. That function will initialize the world state with some built in access control lists. These built in lists are used to bootstrap the environment. So there needs to be access control on who can create resources, groups, and ACLs. If the identify is nil, then use the caller's identify. After the bootstrap is done, the following entities are created: • A resource named ".Resources". A corresponding ACL named ".Resources.ACL" will be created with a single identity pattern in it of the form "%CN%bob.smith@oracle.com", using the actual common name, and the access will be CREATE, READ, UPDATE, and DELETE access. • A group named ".Groups". A corresponding ACL named ".Groups.ACL" will be created with a single identity pattern in it of the form "%CN%bob.smith@oracle.com", using the actual common name, and the access will be CREATE, READ, UPDATE, and DELETE access. • An ACL named ".ACLs". A corresponding ACL control list named ".ACLs.ACL" will be created with a single identity pattern in it of the form "%CN%bob.smith@oracle.com", using the actual common name, and the access will be CREATE, READ, UPDATE, and DELETE access.
NewGroupManager(identity *x509.Certificate, stub shim.ChaincodeStubInterface) (*GroupManager, error)	Get the group manager that's used for all group related operations. Identity: the default identity for related operation. If it's nil, then use caller's identity.
NewACLManager(identity *x509.Certificate, stub shim.ChaincodeStubInterface) (*ACLManager, error)	Get the ACL manager that's used for all ACL related operations. Identity: the default identity for related operation. If it's nil, then use caller's identity.
NewResourceManager(identity *x509.Certificate, stub shim.ChaincodeStubInterface) (*ResourceManager, error)	Get the resource manager that's used for all resource related operations. Identity: the default identity for related operation. If it's nil, then use caller's identity.

Access Control List (ACL) Functions

Definition of ACL structure:

```
type ACL struct {
  Name string
  Description string
  Accesses []string  // CREATE, READ, UPDATE, and DELETE, or whatever 
the end-user defined
  Patterns []string    // identities
  Allowed bool          // true means allows access.
  BindACLs []string  // The list of ACL , control who can call the APIs 
of this struct
}
```

- **Accesses:** The Accesses string is a list of comma-separated arbitrary access names and completely up to the application except for four: CREATE, READ, UPDATE, and DELETE. These access values are used in maintaining the fine grained access control. Applications can use their own access strings such as `"register"`, `"invoke"`, or `"query"`, or even such things as access to field names such as `"owner"`, `"quantity"`, and so on.
- **Allowed:** Allowed determines whether identities that match a pattern are allowed access (true) or prohibited access (false). You could have an access control list that indicates Bob has access to `"CREATE"`, and another one that indicates group Oracle (of which Bob is a member) is prohibited from `"CREATE"`. Whether Bob has access or not depends upon the order of the access control lists associated with the entity in question.
- **BindACLs:** The BindACLs parameter will form the initial access control list.

ACL functions:

Function	Description
Create(acl ACL, identity *x509.Certificate) (error)	Creates a new ACL. Duplicate named ACL are not allowed. To create a new ACL, the identity needs to have CREATE access to the bootstrap resource named ".ACLs". If identity is nil, the default identity specified in newACLManager() is used.
Get(aclName string, identity *x509.Certificate) (ACL, error)	Get a named ACL. The identity must have READ access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.
Delete(aclName string, identity *x509.Certificate) (error)	Delete a specified ACL. The identity must have DELETE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.

Function	Description
Update(acl ACL, identity *x509.Certificate) (error)	Update an ACL. The identity must have UPDATE access to the named resource, and the named ACL must exist. If identity is nil, the default identity specified in NewACLManager() is used.
AddPattern(aclName string, pattern string, identity *x509.Certificate) (error)	Adds a new identity pattern to the named ACL. The identity must have UPDATE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.
RemovePattern(aclName string, pattern string, identity *X509Certificate) (error)	Removes the identity pattern from the ACL. The identity must have UPDATE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.
AddAccess(aclname string, access string, identity *X509Certificate) (error)	Adds a new access to the named ACL. The identity must have UPDATE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.
RemoveAccess(aclName string, access string, identity *X509Certificate) (error)	Removes the access from the ACL. The identity must have UPDATE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.
UpdateDescription(aclName string, newDescription string, identity *X509Certificate) (error)	Update the description. The identity must have UPDATE access to the named ACL. If identity is nil, the default identity specified in newACLManager() is used.
AddBeforeACL(aclName string, beforeName string, newBindACL string, identity *X509Certificate) (error)	Adds a bind ACL before the existing named ACL. If the named ACL is empty or not found, the ACL is added to the beginning of the bind ACL list. The identity must have UPDATE access to the named ACL. If the identity is nil, the default identity specified in newACLManager() is used.
AddAfterACL(aclName string, afterName string, newBindACL string, identity *X509Certificate) (error)	Adds a bind ACL after the existing named ACL. If the named ACL is empty or not found, the ACL is added to the end of the bind ACL list. The identity must have UPDATE access to the named ACL. If the identity is nil, the default identity specified in newACLManager() is used.
RemoveBindACL(aclName string, removeName string, identity *X509Certificate) (error)	Removes the removeName ACL from the bind ACL list. The identity must have UPDATE access to the named ACL. If the identity is nil, the default identity specified in newACLManager() is used.

Function	Description
GetAll(identity *x509.Certificate) ([]ACL, error)	Get all the ACLs. The identity must have READ access to the named ACL. If the identity is nil, the default identity specified in newACLManager() is used.

Group Functions

Definition of `Group` structure:

```
type Group struct {
    Name string
    Description string
    Members []string     // identity patterns, except GRP.
    BindACLs []string    // The list of ACLs, controls who can access
this group.
}
```

Definition of `GroupManager` functions:

Function	Description
Create(group Group, identity *x509.Certificate) (error)	Create a new group. The identity must have CREATE access to bootstrap group ".Group". If identity is nil, the default identity specified in NewGroupManager() is used.
Get(groupName string, identity *x509.Certificate) (Group, error)	Get a specified group. The identity must have READ access to this group. If identity is nil, the default identity specified in NewGroupManager() is used.
Delete(groupName string, identity *x509.Certificate) (error)	Delete a specified group. The identity must have DELETE access to this group. If identity is nil, the default identity specified in NewGroupManager () is used.
AddMembers(groupName string, member []string, identity *x509.Certificate) (error)	Add one or more members into the group. The identity must have UPDATE access to this group. If identity is nil, the default identity specified in NewGroupManager () is used.
RemoveMembers(groupName string, member []string, identity *x509.Certificate) (error)	Remove one or more member from a group. The identity must have UPDATE access to this group. If identity is nil, the default identity specified in NewGroupManager () is used.
UpdateDescription(groupName string, newDes string, identity *x509.Certificate) (error)	Update the description. The identity must have UPDATE access to this group. If identity is nil, the default identity specified in NewGroupManager () is used.

Function	Description
AddBeforeACL(groupName string, beforeName string, aclName string, identity *x509.Certificate) (error)	Adds an bind ACL to the group before the existing named ACL. If the named ACL is empty or not found, the ACL is added to the beginning of the list of bind ACL for the resource. The identity must have UPDATE access to the named group. If identity is nil, the default identity specified in NewGroupManager () is used.
AddAfterACL(groupName string, afterName string, aclName string, identity *x509.Certificate) (error)	Adds a bind ACL to the group after the existing named ACL. If the named ACL is empty or not found, the ACL is added to the end of the list of bind ACL for the group. The identity must have UPDATE access to the named group. If the identity is nil, the default identity specified in NewGroupManager () is used.
RemoveBindACL(groupName string, aclName string, identity *x509.Certificate) (error)	Removes the named ACL from the bind ACL list of the named group. The identity must have UPDATE access to the named group. If the identity is nil, the default identity specified in NewGroupManager () is used.
GetAll(identity *x509.Certificate) ([]Group, error)	Get all groups. The identity must have READ access to these groups. If identity is nil, the default identity specified in NewGroupManager () is used.

Resource Functions

Definition of `Resource` structure:

```
type Resource struct {
     Name string
     Description string
     BindACLs []string      // The name list of ACL, controls who can
access this resource
}
```

Resource Functions:

Fuction	Description
Create(resource Resource, identity *x509.Certificate) (error)	Create a new resource. Duplicate named resources are not allowed. The identity needs to have CREATE access to the bootstrap resource named ".Resources" If identity is null, the default identity specified in NewResourceManager() is used.

Fuction	Description
Get(resName string, identity *x509.Certificate) (Resource, error)	Get a specified resource. The identity must have READ access to the resource. If identity is null, the default identity specified in NewResourceManager() is used.
Delete(resName string, identity *x509.Certificate) (error)	Delete a named resource. The identity must have DELETE access to the named resource. If identity is null, the default identity specified in NewResourceManager() is used.
UpdateDescription(resourceName string, newDes string, identity *x509.Certificate) (error)	Update the description. The identity must have UPDATE access to this resource. If identity is nil, the default identity specified in NewResourceManager() is used.
AddBeforeACL(resourceName string, beforeName string, aclName string, identity *x509.Certificate) (error)	Adds a bind ACL to the resource before the existing named ACL. If the named ACL is empty or not found, the ACL is added to the beginning of the list of bind ACL for the resource. The identity must have UPDATE access to the named resource. If identity is nil, the default identity specified in NewResourceManager() is used.
AddAfterACL(resourceName string, afterName string, aclName string, identity *x509.Certificate) (error)	Adds a bind ACL to the resource after the existing named ACL. If the named ACL is empty or not found, the ACL is added to the end of the list of bind ACL for the resource. The identity must have UPDATE access to the named resource. If the identity is nil, the default identity specified in NewResourceManager() is used.
RemoveBindACL(resourceName string, aclName string, identity *x509.Certificate) (error)	Removes the named ACL from the bind ACL list of the named resource. The identity must have UPDATE access to the named resource. If the identity is nil, the default identity specified in NewResourceManager() is used.
CheckAccess(resName string, access string, identity *x509.Certificate) (bool, error)	Check whether the current user has the specified access to the named resource. If the identity is nil, the default identity specified in NewResourceManager() is used.
GetAll(identity *x509.Certificate) ([]Resource, error)	Get all resources. The identity must have READ access to these resources. If identity is nil, the default identity specified in NewResourceManager() is used.

Example Walkthough Using the Fine-Grained Access Control Library

This topic provides some examples of how this library and chaincode can be used. These all assuming `Init()` has been called to create the bootstrap entities and the caller of `Init()` and `invoke()` is `"%CN%frank.thomas@example.com"`. The normal flow in an application will be to create some initial access control lists that will be used to grant or deny access to the other entities.

Initialization

Call `Initialization()` to create bootstrap entities when instantiating chaincode. For example:

```
import "chaincodeACL"
func (t \*SimpleChaincode) Init(nil, stub shim.ChaincodeStubInterface)
pb.Response
{
         err := chaincodeACL.Initialization(stub)
}
```

Create a new ACL

```
import "chaincodeACL"
...
{

**ACLMgr**  := chaincodeACL.NewACLManager(nil, stub) // Not specify
identity, use caller's identity as default.

// Define a new ACL
**newACL**  := chaincodeACL.ACL{

    "AllowAdmins",   // ACL name
    "Allow admins full access",  // Description
    []string{"CREATE","READ","UPDATE","DELETE"},    // Accesses allowed
or not
    true, // Allowed

[]string{"%CN%bob.dole@example.com","%OU%example.com,"%GRP%admins"}, //
Initial identity patterns
    ".ACLs.acl", // Start with bootstrap ACL

}

// Add this ACL with default identity (caller's identify here)
err :=  **ACLMgr**.Create( **newACL** , nil)

}
```

Now that we have a new ACL, we can use that to modify who can perform certain operations. So we'll first add this new ACL to the bootstrap group `.Groups` to allow any admin to create a group.

Add an ACL to a group

```
import "chaincodeACL"
…
{

  **groupMgr**  := chaincodeACL.NewGroupManager(nil, stub) // Not
specify identity, use caller's identity as default.
  err :=  **groupMgr**.AddAfterACL(

    ".Groups",     // Bootstrap group name
    ".Groups.ACL", // Which ACL to add after
    "AllowAdmins", // The new ACL to add
    nil            // with default identity that's frank.thomas

)

}
```

This adds the `AllowAdmins` ACL to the bootstrap group `.Groups` after the initial bootstrap ACL. Thus this ensures that Frank Thomas can still perform operations on `.Groups` as the ACL granting him permission is first in the list. But now anyone that matches the `AllowAdmins` ACL can perform CREATE, READ, UPDATE, or DELETE operations (they can now create new groups).

Create a new group

Admins can now create a new group:

```
import "chaincodeACL"
...
{

...
  // Define a new group.
  **newGroup**  := chaincodeACL.Group{

      "AdminGrp",   // Name of the group
      "Administrators of the app",   // Description of the group

{"%CN%jill.muller@example.com","%CN%ivan.novak@example.com","%ATTR%role=
admin"},
      []string{"AllowAdmins"},   // The ACL for the group

    }

  **groupMgr**  := chaincodeACL.NewGroupManager(nil, stub)   // Not
specify identity, use caller's identity as default.
  err :=  **groupMgr**.Create( **newGroup** ,
bob\_garcia\_certificate)   // Using a specific certificate
...
}
```

This call is using an explicit identity - that of Bob Garcia (using his certificate) - to try and create a new group. Since Bob Garcia matches a pattern in the `AllowAdmins` ACL and members of that ACL can perform CREATE operations on the bootstrap group `.Groups`, this call will succeed. Had Jim Silva - who was not in organization unit `example.com` nor in the group `AdminGrp` (which still doesn't exist) - had his certificate passed as the last argument, the call would fail as he doesn't have the appropriate permissions. This call will create a new group called "`AdminGrp`" with initial members of the group being jill.muller@example.com and ivan.novak@example.com or anyone with the attribute (ABAC) role=admin.

Create a new resource

```
import "chaincodeACL"
...
{

  ...
  **newResource**  :=  **chaincodeACL**.Resource{

      "transferMarble", // Name of resource to create

      "The transferMarble chaincode function", // Description of the
resource

      []string{"AllowAdmins"}, // Single ACL for now allowing admins

  }

  **resourceMgr**  :=  **chaincodeACL**.NewResourceManager(nil,
stub)  // Not specify identity, use caller's identity as default.
  err :=  **resourceMgr**.Create(resourceMgr, nil)   // Using caller's
certificate

  ...
}
```

This would create a new resource named `transferMarble` that the application might use to control access to the `transferMarble` chaincode function. The access is currently limited by the `AllowAdmins` access control list.

Check access for a resource

We can use this new resource in our chaincode to only allow admins to transfer a marble by modifying the `invoke()` method of the Marbles chaincode as follows:

```
import "chaincodeACL"
…
func (t \*SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface)
pb.Response {
  **resourceMgr**  :=  **chaincodeACL**.NewResourceManager(nil, 
stub)   // Not specify identity, use caller's identity as default.

  function, args := stub.GetFunctionAndParameters()

  fmt.Println("invoke is running " + function)        // Handle 
different functions

  if function == "initMarble" {   //create a new marble

      return t.initMarble(stub, args)}

  else if function == " **transferMarble**" { //change owner of a 
specific marble

    **allowed** , err : =  **resourceMgr**. **CheckAccess**
("transferMarble", "UPDATE", nil)
    if  **allowed**  == true {

      return t.transferMarble(stub, args)

    else {

      return NOACCESS

    }

    } else if function == "transferMarblesBasedOnColor" { //transfer 
all marbles of a certain color
    …

    }

}
```

Fine-Grained Access Control Marbles Sample

The marbles chaincode application lets you create assets (marbles) with unique attributes (name, size, color and owner) and trade these assets with fellow participants in a blockchain network.

This sample application includes a variety of functions to let you examine how to work with access control lists and groups to restrict functions to certain users.

- Overview of the Sample
- Pre-requisites and Setup
- Implement the Fine-Grained Access Control Marble Sample
- Testing the Access Control
- Sample Files Reference

Overview of the Sample

The test scenario included in the sample contains the following restrictions in order to manage assets:

- Bulk transfer of red marbles is only allowed by identities having the "`redMarblesTransferPermission`" Fabric attribute.
- Bulk transfer of blue marbles is only allowed by identities having the "`blueMarblesTransferPermission`" Fabric attribute.
- Deletion of marbles is only allowed to identities with "`deleteMarblePermission`" Fabric attribute.

These restrictions are enforced by implementing the following library methods in the `fgMarbles_chaincode.go` chaincode:

- Create a fine-grained ACL group named `bulkMarblesTransferGroup`. This group will define all the identities which can transfer marbles based on color (bulk transfers):

  ```
  createGroup(stub, []string{" bulkMarblesTransferGroup",
  "List of Identities allowed to Transfer Marbles in Bulk",
  "%ATTR%redMarblesTransferPermission=true,
  %ATTR%blueMarblesTransferPermission=true", ".ACLs"})
  ```

- Create a fine-grained ACL named `redMarblesAcl` which provides bulk transfer of red marbles access to `bulkMarblesTransferGroup`:

  ```
  createACL(stub, []string{"redMarblesAcl",
  "ACL to control who can transfer red marbles in bulk",
  "redMarblesTransferPermission", "%GRP%bulkMarblesTransferGroup",
  "true", ".ACLs"})
  ```

- Create a fine-grained ACL named `blueMarblesAcl` which provides bulk transfer of blue marbles access to `bulkMarblesTransferGroup`:

  ```
  createACL(stub, []string{"blueMarblesAcl",
  "ACL to control who can transfer blue marbles in bulk",
  "blueMarblesTransferPermission", "%GRP%bulkMarblesTransferGroup",
  "true", ".ACLs"})
  ```

- Create a fine-grained ACL named `deleteMarbleAcl` to restrict marble deletion based on "`canDeleteMarble=true`" Fabric attribute:

  ```
  createACL(stub, []string{"deleteMarbleAcl",
  "ACL to control who can Delete a Marble",
  "deleteMarblePermission", "%ATTR%deleteMarblePermission=true",
  "true", ".ACLs"})
  ```

- Create a fine-grained ACL resource named `marble`, operations on which are controlled using the various ACLs we created:

  ```
  createResource(stub, []string{"marble",
  "System marble resource",
  "deleteMarbleAcl,blueMarblesAcl,redMarblesAcl,.ACLs"})
  ```

Pre-requisites and Setup

In order to run the fine-grained access control version of the marbles sample, complete these steps:

1. Download the fine-grained access control version of the marbles sample. On the **Developer Tools** tab, open the **Samples** pane, and then click the download link under **Marbles with Fine-Grained ACLs**. Unzip this package - it contains zips of the marbles sample (`fgACL_MarbleSampleCC.zip`), Node.js files to run the sample (`fgACL-NodeJSCode.zip`), and the fine-grained access control library (`Fine-GrainedAccessControlLibrary.zip`).
2. Generate the chaincode package that will be deployed to Blockchain Platform:
   - Install govendor:

     ```
     go get -u github.com/kardianos/govendor
     ```

   - Unzip the contents of `fgACL_MarbleSampleCC.zip` to the `fgACL_MarbleSampleCC` directory. The contents of the `fgACL_MarbleSampleCC` directory would be: `fgACL_Operations.go`, `fgGroups_Operations.go`, `fgMarbles_chaincode.go`, `fgResource_Operations.go` and the `vendor` directory.
   - From a command line, go to the `fgACL_MarbleSampleCC` directory, and run `govendor sync`. This will download the required dependency (`github.com/op/go-logging`) and add it to the `vendor` directory.
   - Zip all the contents (the four Go files and the `vendor` directory) of the `fgACL_MarbleSampleCC` directory. Your chaincode is ready to be deployed to Blockchain Platform.
3. Install and instantiate the updated sample chaincode package (`fgACL_MarbleSampleCC.zip`) as described in Use Quick Deployment.
4. On the **Developer Tools** tab, open the **Application Development** pane, and then follow the instructions to download the Node.js SDK.
5. On the **Developer Tools** tab, open the **Application Development** pane, and then click **Download the development package**.
   a. Unzip the development package into the same folder with the Node.js files downloaded with the sample.
   b. In the `network.yaml` file, look for the `certificateAuthorities` entry and its `registrar` entry. The administrator's password is masked (converted to `***`) in the `network.yaml` when downloaded. It should be replaced with the administrator's clear text password when running this sample.
6. Register a new identity with your Blockchain Platform instance:

a. Create a new user in IDCS (referred to as `<NewIdentity>` in the following steps) in the IDCS mapped to your tenancy.
b. Give this user the `CA_User` application role for your instance.

Implement the Fine-Grained Access Control Marble Sample

The following steps will enroll your new user and implement the ACL restrictions using the provided Node.js scripts.

1. **Enroll the new user:**

   ```
   node registerEnrollUser.js <NewIdentity> <Password>
   ```

2. **Initialization:** Initialize the access control lists.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> ACLInitialization
   ```

3. **Create the access control lists, groups, and resources:** This creates the ACL resources described in the overview.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> createFineGrainedAclSampleResources
   ```

4. **Create your test marble resources:** This creates several test marble assets - blue1 and blue2 owned by tom, red1 and red2 owned by jerry, and green1 and green2 owned by spike.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> createTestMarbles
   ```

Testing the Access Control

In order to test that our access control lists are only allowing the correct users to perform each function, we'll run through some sample scenarios.

1. **Transfer a marble:** We're transferring marble `blue1` from tom to jerry. Since there are no restrictions on who can transfer a single marble, this should complete successfully.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarble blue1 jerry
   ```

2. **Transfer a marble as the administrative user:** We're transferring marble `blue1` from jerry to spike. Since there are no restrictions on who can transfer a single marble, this should also complete successfully.

   ```
   node invokeQueryCC.js <AdminIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarble blue1 spike
   ```

3. **Get history:** Now we'll query the history of the marble named `blue1`. It should return that it was transferred first to jerry then to spike.

   ```
   node invokeQueryCC.js <AdminIdentity> <Password> <ChannelName>
   <ChaincodeName> getHistoryForMarble blue1
   ```

4. **Transfer all red marbles:** The `redMarblesAcl` ACL should allow this transfer because the newly registered identity has the required `"redMarblesTransferPermission=true"` Fabric attribute, so the two red marbles should be transferred to tom.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarblesBasedOnColor red tom
   ```

5. **Transfer all red marbles as the administrative user:** The administrative identity doesn't have the `"redMarblesTransferPermission=true"` Fabric attribute, so the `redMarblesAcl` ACL should block this transfer.

   ```
   node invokeQueryCC.js <AdminIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarblesBasedOnColor red jerry
   ```

6. **Transfer all green marbles:** By default, only explicitly defined access is allowed. Because there isn't an ACL which allows for bulk transfer of green marbles, this should fail.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> transferMarblesBasedOnColor green tom
   ```

7. **Delete a marble:** The `deleteMarbleAcl` ACL allows this deletion because the newly registered identity has the required `"deleteMarblePermission=true"` Fabric attribute.

   ```
   node invokeQueryCC.js <NewIdentity> <Password> <ChannelName>
   <ChaincodeName> delete green1
   ```

8. **Delete a marble as the administrative user:** The `deleteMarbleAcl` ACL should prevent this deletion because the administrative identity doesn't have the required `"deleteMarblePermission=true"` Fabric attribute.

   ```
   node invokeQueryCC.js < AdminIdentity > <Password> <ChannelName>
   <ChaincodeName> delete green2
   ```

Sample Files Reference

These tables list the methods available in the chaincode and application files included with the sample.

fgMarbles_chaincode.go

Function	Description
`initMarble`	Create a new marble
`transferMarble`	Transfer a marble from one owner to another based on name

Function	Description
`createTestMarbles`	Calls `initMarble` to create new sample marbles for testing purposes
`createFineGrainedAclSampleResources`	Creates the fine-grained access control list (ACL), groups, and resources required by our test scenario
`transferMarblesBasedOnColor`	Transfers multiple marbles of a certain color to another owner
`delete`	Delete a marble
`readMarble`	Returns all attributes of a marble based on name
`getHistoryForMarble`	Returns a history of values for a marble

fgACL_Operations.go

Methods	Parameters	Description
`getACL`	• `name`	Get a named ACL or read all ACLs. The user invoking the method must have READ access to the named ACL.
`createACL`	• `name` • `description` • `accesses` • `patterns` • `allowed` • `BindACLs` • `Identity_Certificate`	To create a new ACL, the user invoking the method needs to have CREATE access to the bootstrap resource named `".ACLs"`. Duplicate named ACLs are not allowed
`deleteACL`	• `name`	The user invoking the method must have DELETE access to the named ACL.
`updateACL`	• `name` • `description` • `accesses` • `patterns` • `allowed` • `BindACLs`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`addAfterACL`	• `aclName` • `existingBindAclName` • `newBindAclName`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`addBeforeACL`	• `aclName` • `existingBindAclName` • `newBindAclName`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`addPatternToACL`	• `aclName` • `BindPattern`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.

Methods	Parameters	Description
`removePatternFromACL`	• `aclName` • `BindPattern`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`updateDescription`	• `aclName` • `newDesc`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`removeBindACL`	• `aclName` • `bindAclNameToRemove`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`addAccess`	• `aclName` • `accessName`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`removeAccess`	• `aclName` • `accessName`	The user invoking the method must have UPDATE access to the named resource, and the named ACL must exist.
`ACLInitialization`	• none	This function is used to initialize the fine-grained ACL support.

fgGroups_Operations.go

Methods	Parameters	Description
`getGroup`	• `name`	If name="`GetAll`", it returns all the groups the identity has access to. Otherwise, it returns the individual group details (if accessible) based on name. The user invoking the method must have READ access to this group.
`createGroup`	• `name` • `description` • `patterns` • `bindACLs`	Returns `success` or `error`. The user invoking the method must have CREATE access to bootstrap group "`. Group`"
`deleteGroup`	• `name`	The user invoking the method must have DELETE access to this group.
`addAfterGroup`	• `groupName` • `existingBindAclName` • `newBindAclName`	The user invoking the method must have UPDATE access to this group.
`addBeforeGroup`	• `groupName` • `existingBindAclName` • `newBindAclName`	The user invoking the method must have UPDATE access to this group.

Methods	Parameters	Description
`updateDescriptionForGroup`	• `groupName` • `newDesc`	The user invoking the method must have UPDATE access to this group.
`removeBindAclFromGroup`	• `groupName` • `bindAclNameToRemove`	The user invoking the method must have UPDATE access to this group.
`addMembersToGroup`	• `groupName` • `pattern`	The user invoking the method must have UPDATE access to this group.
`removeMembersFromGroup`	• `groupName` • `pattern`	The user invoking the method must have UPDATE access to this group.

fgResource_Operations.go

Methods	Parameters	Description
`createResource`	• `name` • `description` • `bindACLs`	The user invoking the method needs to have CREATE access to the bootstrap resource named `". Resources"`. Duplicate named resources are not allowed.
`getResource`	• `name`	The user invoking the method must have READ access to the resource
`deleteResource`	• `name`	The user invoking the method must have DELETE access to the named resource
`addAfterACLInResource`	• `ResourceName` • `existingBindAclName` • `newBindAclName`	The user invoking the method must have UPDATE access to this resource
`addBeforeACLInResource`	• `ResourceName` • `existingBindAclName` • `newBindAclName`	The user invoking the method must have UPDATE access to this resource
`updateDescriptionInResource`	• `ResourceName` • `newDesc`	The user invoking the method must have UPDATE access to this resource
`removeBindACLInResource`	• `ResourceName` • `bindAclNameToRemove`	The user invoking the method must have UPDATE access to this resource
`checkResourceAccess`	• `ResourceName` • `access`	Checks whether the current user invoking the method has the specified access to the named resource.

C

Using Blockchain App Builder for Oracle Blockchain Platform

Blockchain App Builder for Oracle Blockchain Platform is a tool set that assists with rapid development, testing, debugging, and deployment of chaincode on Oracle Blockchain Platform networks, comprising cloud BaaS nodes on Oracle Cloud Infrastructure and/or on-premises nodes using Enterprise Edition.

A smart contract (also known as a chaincode) defines the different states of a business object between two or more parties and business logic that validates and implements changes as the object moves between these different states. At the heart of every blockchain application is one or more chaincodes. So a chaincode should be bug-free and tested before it is deployed and instantiated.

You can use Blockchain App Builder to generate complex chaincodes from a simple configuration file and assets specification in TypeScript (for node.js chaincode) and Go (for golang chaincode) from a simple specification file. With the specification file you can specify multiple asset definitions and behaviors. You can then generate and test your chaincodes either on your local machine by using a preconfigured instance of Hyperledger Fabric inside Blockchain App Builder, or by connecting to your Oracle Blockchain Platform network.

Note:

Although JavaScript isn't supported by Blockchain App Builder, because the TypeScript projects are compiled to JavaScript, you can add basic JavaScript to a TypeScript project if needed.

The Blockchain App Builder supports the full development lifecycle either using a command line interface or as an extension for Visual Studio Code.

To get the Blockchain App Builder tools and samples, in the console open the **Developer Tools** tab and select the **Blockchain App Builder** pane. From here you can download the command line interface tools or the Visual Studio Code extension. Additionally, there are two samples - Fabcar and Marbles - which can be used to see how the tools work or as a template for your own chaincode projects.

Topics:

Using the Blockchain App Builder Command Line Interface

The Blockchain App Builder command line interface helps you build and scaffold a fully-functional chaincode project from a specification file. Once the project is built, you can run and test it on a local Hyperledger Fabric network, or your provisioned

Oracle Blockchain Platform network. You can then run SQL rich queries, debug the chaincode, or write and run unit tests using the generated code.

Table C-1 Workflow When Using the CLI

Task	Description	Related Topics
Install and configure	Download the Blockchain App Builder CLI from your Oracle Blockchain Platform console and install it and any prerequisite software.	• Install and Configure Blockchain App Builder CLI
Create the chaincode project	Create a specification file, and then run the CLI initialization process to generate your chaincode project from that file.	• Create a Chaincode Project with the Blockchain App Builder CLI Detailed reference information about the structure and contents of the specification file and the generated chaincode project: • Input Specification File • Scaffolded TypeScript Chaincode Project • Scaffolded Go Chaincode Project
Deploy the chaincode	After your chaincode project is created, you can deploy it locally to the included pre-configured Hyperledger Fabric network, or remotely to your Oracle Blockchain Platform Cloud or Enterprise Edition. You can also package the chaincode project for manual deployment to Oracle Blockchain Platform.	• Deploy Your Chaincode to a Local Hyperledger Fabric Network • Deploy Your Chaincode to a Remote Oracle Blockchain Platform Network • Package Your Chaincode Project for Manual Deployment to Oracle Blockchain Platform
Test the chaincode	Once your chaincode is running on a network, you can test any of the generated methods. Additionally, If you chose to create the `executeQuery` method during your chaincode development, you can run SQL rich queries if your chaincode is deployed to an Oracle Blockchain Platform network.	• Test Your Chaincode on a Local Hyperledger Fabric Network • Test Your Chaincode on a Remote Oracle Blockchain Platform Network • Execute Berkeley DB SQL Rich Queries
Debug the chaincode	The Blockchain App Builder extension for Visual Studio Code includes line-by-line debugging of your chaincode.	• Debugging from Visual Studio Code
Synchronize your updates	When you update your specification file, you can synchronize the changes with the generated chaincode files.	• Synchronize Specification File Changes With Generated Source Code
Running unit tests	A basic unit test case setup is included in the project. Additional tests can be added and run.	• Writing Unit Test Cases and Coverage Reports for the Chaincode Project

Install and Configure Blockchain App Builder CLI

The following platforms are supported:

- Mac OSX
- Oracle Linux 7.7 or 7.8
- Windows 10

Once you've completed the install:

- Verify your installation.
- If you're using Go chaincode projects, complete the additional configuration steps.

Prerequisites for Mac OSX and Linux

Before you install Blockchain App Builder CLI on your local system, you must install the prerequisites.

- Docker v18.09.0 or later
- Docker Compose v1.23.0 or later
- Node v10.18.1 or later (tested with 10.22.1)
- npm v6.x (tested with 6.13.4)
- If you are developing Go chaincodes, you need to install Go v1.14
- If you want to use the Synchronization feature of App Builder, Git should be installed and your username and email should be configured as follows.

  ```
  > git config --global user.name "<your_name>"
  > git config --global user.email "<email>"
  ```

You also need to download the Blockchain App Builder CLI package (`oracle-ochain-cli-1.0.0.tgz`) from your Oracle Blockchain Platform console. It can be found on the **Developer Tools** tab on the **Blockchain App Builder** pane.

Install Node and npm using nvm

We suggest using nvm to install Node and npm because it will give you the ability to run more commands without sudo.

1. Install nvm:

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/
   install.sh | bash
   ```

2. Add the below code snippet to `~/.bash_profile`, `~/.profile`, `~/.bashrc` or `~/.zshrc`.

   ```
   export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "$
   {HOME}/.nvm" || printf
   %s "${XDG_CONFIG_HOME}/nvm")"
   [ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
   ```

3. Log out and log back in to your operating system.

4. Verify the nvm installation:

   ```
   nvm version
   ```

5. Install Node and npm:

   ```
   nvm install 10.18.1
   ```

6. Set Node 10.18.1 as the default in nvm:

   ```
   nvm alias default 10.18.1
   ```

Install Blockchain App Builder on Mac OSX or Linux

Mac OSX

1. Install Xcode or the XCode command line tools (xcode-select).

   ```
   sudo xcode-select –install
   ```

2. Install Blockchain App Builder:

   ```
   npm install -g oracle-ochain-cli-1.4.0.tgz
   ```

Note that Mac OS Catalina can have issues with xcode-select. If this happens, reset and restart:

```
xcode-select –-reset
```

Oracle Enterprise Linux

1. Ensure that yum is updated and pointing to the current repository based on your kernel. See the following two articles for information on how to do this:
   - https://blogs.oracle.com/virtualization/install-docker-on-oracle-linux-7-v2
   - https://oracle-base.com/articles/linux/download-the-latest-oracle-linux-repo-file#oracle-linux-7-updated
2. Install the required libraries:

   ```
   sudo yum install gcc gcc-c++ docker-engine -y
   ```

3. Ensure the current user has access to Docker:

   ```
   sudo groupadd docker
   sudo usermod -aG docker $USER
   ```

4. Enable the Docker service:

   ```
   sudo systemctl enable docker
   sudo systemctl start docker
   ```

5. Install Blockchain App Builder:

   ```
   npm install -g oracle-ochain-cli-1.4.0.tgz
   ```

6. Log out the current user and log in again for group membership to take effect.

Prerequisites for Windows

Before you install Blockchain App Builder CLI on your local system, you must install the prerequisites.

- Docker Desktop for Windows v2.x (tested with 2.5.0.1). When prompted by Docker, provide the Filesharing permissions (required for App Builder).
- Node v10.18.1 or later (tested with 10.22.1)
- npm v6.x (tested with 6.13.4)
- Perl v5.x (tested with ActiveState Perl 5.28)
- Install Windows Build Tools in a powershell with administrative access. `npm install --global windows-build-tools`
- If you are developing Go smart contracts, install Go v1.14
- If you want to use the Synchronization feature of App Builder, Git should be installed and your username and email should be configured as follows.

  ```
  > git config --global user.name "<your_name>"
  > git config --global user.email "<email>"
  ```

- Download and build OpenSSL

Download and Build OpenSSL

1. Download OpenSSL from: https://www.openssl.org/source/old/1.0.2/openssl-1.0.2u.tar.gz
2. Unzip the tarball.
3. Open the Visual C++ 2017/2019 Native Tools command prompt. In the Windows search bar, search for `x64 Native Tools Command Prompt for VS`.
4. Navigate to the extracted OpenSSL folder. Run the following commands as an administrator:

   ```
   > perl Configure VC-WIN64A –prefix=C:\OpenSSL-Win64
   > ms\do_win64a
   > nmake -f ms\ntdll.mak          This can take up to 15 minutes to
   complete.
   > cd out32dll
   > ..\ms\test
   > cd ..
   > md C:\OpenSSL-Win64
   > md C:\OpenSSL-Win64\bin
   > md C:\OpenSSL-Win64\lib
   > md C:\OpenSSL-Win64\include
   > md C:\OpenSSL-Win64\include\openssl
   > copy /b inc32\openssl\* C:\OpenSSL-Win64\include\openssl
   > copy /b out32dll\ssleay32.lib C:\OpenSSL-Win64\lib
   ```

```
> copy /b out32dll\libeay32.lib C:\OpenSSL-Win64\lib
> copy /b out32dll\ssleay32.dll C:\OpenSSL-Win64\bin
> copy /b out32dll\libeay32.dll C:\OpenSSL-Win64\bin
> copy /b out32dll\openssl.exe C:\OpenSSL-Win64\bin
> copy /b C:\OpenSSL-Win64\bin\libeay32.dll
C:\Windows\System32\libeay32.dll
> copy /b C:\OpenSSL-Win64\bin\ssleay32.dll
C:\Windows\System32\ssleay32.dll
```

Install Blockchain App Builder on Windows

When you've installed all the prerequisite software, install Blockchain App Builder:

```
> npm install -g oracle-ochain-cli-1.x.x.tgz
```

Verify the Install

In your terminal, type `ochain -v`. The output should list the Blockchain App Builder CLI usage, options, and commands.

Additional Setup for Go Chaincode Projects

In order to develop a Go project, you will need to setup the `GOPATH` environment variable. This will allow Go to properly locate your workspace in order to run your code.

Linux and Mac

Before setting your `GOPATH`, make sure you have a `go/` folder in your `$HOME`. If not, create `go/` in your home directory:

```
mkdir $HOME/go
```

Set your `GOPATH` by adding this variables in your `~/.bash_profile`, `~/.profile`, `~/.bashrc` or `~/.zshrc` file.

```
export PATH=$PATH:/usr/local/go/bin
export GOPATH=$HOME/go
export PATH=$PATH:$GOPATH/bin
```

After editing, run the following to make your changes take effect immediately:

```
source ~/.bash_profile
```

Alternatively, you can apply this change system wide by adding the above variables in the `/etc/bashrc` file.

Windows

Create `go/` in your home directory: `C:\Users\<username>\go`

Create a Chaincode Project with the Blockchain App Builder CLI

To create a Chaincode Project when using the Blockchain App Builder CLI, you need to scaffold a chaincode project from a detailed specification file. This generates a project with all the files you need.

Background

Blockchain App Builder's `init` command initializes and scaffolds a chaincode project right out of the box for you. Based on simple input, the `init` command can generate complex chaincode projects with features such as:

- Multiple assets (models) and their behaviors (controllers)
- Auto-generate CRUD (Create/Read/Update/Delete) and non-CRUD methods
- Automatic validation of arguments
- Marshalling/unmarshalling of arguments
- Transparent persistence capability (ORM)
- Calling rich queries

The generated project follows model/controller and decorator pattern, which allows an asset's properties maintained on the ledger to be specified as typed fields and extended with specific behaviors and validation rules. This reduces the number of lines of code which helps in readability and scalability.

Prerequisites

Before you begin, you need to create an input specification file. The detailed specifications for this file are described here: Input Specification File.

Scaffolding the Chaincode Project with the `init` Command

Typing `ochain init -h` will list the command usage with all its options. The `init` command has four options:

- **--cc/-c:**
  The name of the chaincode project. The default value is `MyChaincode`.
- **--lang/-l**
  The language of the scaffolded chaincode. Currently Blockchain App Builder supports Typescript (`ts`) and Go (`go`). If no option is given it is defaulted to `ts`.
- **--conf/-f** or **-spec**
  The path to an input specification file. Blockchain App Builder reads the input specification file and generates the scaffolded project with many helper tools which helps in reducing the overall development effort. Taking full advantage of the input specification file can significantly reduce the development time.

  The specification file could be in `yaml` or `json` format. If the path is not specified, it defaults to the current directory. See Input Specification File.
- **--out/-o**
  The output directory of the scaffolded chaincode project. If not specified, the scaffolded project is generated in the current directory.

  The output is a fully contained, light-weight and scalable Typescript or Go chaincode project.

Example

```
my-mac:~ name$ ochain init --cc MyNewTsProject --lang ts --conf spec.yml
```

Defaults

If no options were specified in the `ochain init` command, the name of the scaffolded project is `MyChaincode` and the language is TypeScript.

The `MyChaincode.model.ts` contains only one asset called `MyAsset` with one property named `value`. `MyChaincode.controller.ts` contains one controller with the corresponding CRUD methods for the `MyAsset` model.

Output

When the process is complete, you'll have a fully-functional chaincode project that you can deploy either locally or to a remote Oracle Blockchain Platform. For a detailed overview of the files created, see:

- Scaffolded TypeScript Chaincode Project
- Scaffolded Go Chaincode Project

Input Specification File

The Blockchain App Builder initialization command reads the input specification file and generates the scaffolded project with several tools to assist in the chaincode development process.

With the specification file you can specify multiple asset definitions and behavior, CRUD and non-CRUD method declaration, custom methods, validation of arguments, auto marshalling/unmarshalling, transparent persistence capability, and invoking rich data queries using SQL SELECTs or CouchDB Query Language. These features will be generated for you.

The specification file can be written in either `yaml` or `json`. You can see sample specification files in both formats in the Blockchain App Builder package download:

- `fabcar.yml`
- `marbles.yml`

Note:

As per Go conventions, exported names begin with a capital letter. Therefore all the asset properties and methods must have names starting with capital letters in the specification file.

Structure of the Specification File

The specification file should be structured in the following way:

```
assets:
    name:
    properties:
        name:
        type:
        id:
        derived:
           strategy:
           algorithm:
           format:
        mandatory:
        default:
        validate:
    methods:
        crud:
        others:
    type:
customMethods:
```

Table C-2   Specification File Parameter Descriptions and Examples

Entry	Description	Examples
`assets:`	This property takes the definition and behavior of the asset. You can give multiple asset definitions here.	
`name:`	The name of the asset.	`name: owner # Information about the owner`
`properties:`	Describe all the properties of an asset.	
`name:`	The name of the property.	`name: ownerId # Unique ID for each owner`

`type:`	Basic types are supported:
	• `number`
	• `string`
	• `boolean`
	• `date`
	• `array`
	• `embedded`
	For Go chaincodes, `number` is mapped to `init`. Other types such as
	• `float`
	• `complex`
	• `unsigned/ signed int`
	• `8/16/32/64 bits`
	are not supported at this time.
		`name: year        # Model year`
		`type: number`
		`mandatory: true`
		`validate: min(1910),max(2020)`
		`name: color       # Color - no validation as color names are innumerable`
		`type: string`
		`mandatory: true`

Table C-2 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`id:`	• true
	This specifies the identifier of this asset. This property is mandatory.
		```
		name: owner            # Informmation about the
		owner
		properties:
		     name: ownerId    # Unique ID for each owner
		     type: string
		     mandatory: true
		     id: true
		     name: name        # Name of the owner
		     type: string
		     mandatory: true
		```

Table C-2 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`derived:`	This property specifies that the id property is derived from other keys. Dependent properties should be `string` datatype and not an embedded asset.
	This property has two mandatory parameters:
	• `strategy`: takes values of `concat` or `hash`.
	• `format`: takes an array of specification strings and values to be used by the strategy.
	Example 1:
	• The property `employeeID` is dependent on the `firstName` and `lastName` properties.
	• This property is a concatenation of the values listed in the `format` array.
	• `IND%1#%2%tIND` is the 0th index in the array and describes the final format.
	• `%n` is a position specifier that takes its values from the other indexes in the array.
	• `%t` indicates the value should be `stub.timestamp` from the channel header.
	• If you need to use the character `%` in the `format`
		Example 1
		
		`name: employee`
		`  properties:`
		`     name: employeeId`
		`     type: string`
		`     mandatory: true`
		`     id: true`
		`     derived:`
		`         strategy: concat`
		`         format:`
		`["IND%1#%2%tIND","firstName","lastName"]`
		
		`     name: firstName`
		`     type: string`
		`     validate: max(30)`
		`     mandatory: true`
		
		`     name: lastName`
		`     type: string`
		`     validate: max(30)`
		`     mandatory: true`
		
		`     name: age`
		`     type: number`
		`     validate: positive(),min(18)`
		
		Example 2
		
		`name: account`
		`  properties:`
		`     name: accountId`
		`     type: string`
		`     mandatory: true`
		`     id: true`
		`     derived:`
		`         strategy: hash`
		`         algorithm: 'sha256'`
		`         format:`
		`["IND%1#%2%t","bankName","ifsccode"]`
		
		`     name: bankName`
		`     type: string`
		`     validate: max(30)`
		`     mandatory: true`
		
		`     name: ifsccode`

Table C-2 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

	string, it should be escaped with another `%`.
	• The final format in this example would be: `INDfirstName#lastName1606885454916IND`
	
	Example 2:
	• When using `hash`, you must also use the `algorithm` parameter. The default is `sha256`; `md5` is also supported.
	• `IND%1#%2%t` is the 0th index in the array and describes the final format.
	• `%n` is a position specifier that takes its values from the other indexes in the array.
	• `%t` indicates the value should be `stub.timestamp` from the channel header.
	• If you need to use the character `%` in the `format` string, it should be escaped with another `%`.
		`type: string`
		`mandatory: true`

Table C-2 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`mandatory:`	• true
	• false
	The corresponding property is mandatory and cannot be skipped while creating an asset.
		`name: phone       # Phone number - validate as (ddd)-ddd-dddd where dashes could also be periods or spaces`
		`type: string`
		`mandatory: true`
		`validate: /^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$/`
		`name: cars        # The list of car VINs owned by this owner`
		`type: string[]`
		`mandatory: false`

`default:`	This gives you the default value of this property.	
`validate:`	The given property is validated against some of the out-of-box validations provided by Blockchain App Builder. You can chain validations if you ensure that the chain is valid. If the `validate` property is not provided, then the validation is done against only the property `type`.	

`validate:`
type: number
	• positive()
	• negative()
	• min()
	• max()
	These validations can be chained together separated by commas.
		`name: offerApplied`
		`type: number`
		`validate: negative(),min(-4)`
		
		`name: year   # Model year`
		`type: number`
		`mandatory: true`
		`validate: min(1910),max(2020)`

Table C-2 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`validate:`
type: string
	• `min()`
	• `max()`
	• `email()`
	• `url()`
	• `/regex/` - supports PHP regex
	For Go chaincodes, regular expressions which contain certain reserved characters or whitespace characters should be properly escaped.
		`name: website`
		`type: string`
		`mandatory: false`
		`validate: url()`
		
		`name: phone  # Phone number - validate as (ddd)-ddd-dddd where dashes could also be periods or spaces`
		`type: string`
		`mandatory: true`
		`validate: /^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$/`
		
		`name: Color #Color can be red, blue, or green`
		`type: string`
		`mandatory: true`
		`validate: /^\\s*(red|blue|green)\\s*$/`

`validate:`
type: boolean
	• true
	• false
	In the example, the validation of property `active` is by the type itself (`boolean`)
		`name: active`
		`type: boolean`

`validate:`
type: array
	By type itself, in the form of `type: number[]`, this conveys that the array is of type number.
	You can enter limits to the array in the format `number[1:5]` which means minimum length is 1, maximum is 5. If either one is avoided, only min/max is considered.
		`name: items`
		`type: number[:5]`

Table C-2 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`validate:`
type: date
	• `min()`
	• `max()`
	Date should be one of these formats:
	• `YYYY-MM-DD`
	• `YYYY-MM-DDTHH:MM:SSZ`, where T separates the date from the time, and the Z indicates UTC. Timezone offsets can replace the Z as in -05:00 for Central Daylight Savings Time.
		`name: expiryDate`
		`type: date`
		`validate: max('2020-06-26')`
		
		`name: completionDate`
		`type: date`
		`validate: min('2020-06-26T02:30:55Z')`

`methods:`	Use this to state which of the CRUD (Create/Read/Update/Delete) or additional methods are to be generated.
	By default, if nothing is entered, all CRUD and other methods are generated.
		`methods:`
		`    crud: [create, getById, update, delete]`
		`    others: [getHistoryById, getByRange]`

`crud:`	• `create`
	• `getByID` (read)
	• `update`
	• `delete`
	If this array is left empty, no CRUD methods will be created.
	If the `crud` parameter is not used at all, all four methods will be created by default.
		`methods:`
		`    crud: [create, getById, delete]`
		`    others: [] # no other methods will be created`

Table C-2 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`others:`	• `getHistoryById`
	• `getByRange`
	`getHistoryById` returns the history of the asset in a list.
	`getByRange` returns all the assets in a given range.
	If this array is left empty, no other methods will be created.
	If the `others` parameter is not used at all, both methods will be created by default.
		```
		methods:
		    crud: [create, delete]
		    others: [] # no other methods will be created
		```
		```
		methods:
		  crud: [create, getById, update, delete]
		  others: [getHistoryById, getByRange]
		```

Table C-2 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`type:`	This attribute if set to `embedded` defines the asset as an embedded asset. Embedded assets do not have CRUD methods and have to be part of another asset to store in the ledger.
	
	In the example, the property `address` is embedded, and is defined in another asset.
		Asset: `employee`
		
		`name: employee`
		  `properties:`
		     `name: employeeId`
		     `type: string`
		     `mandatory: true`
		     `id: true`
		
		     `name: firstName`
		     `type: string`
		     `validate: max(30)`
		     `mandatory: true`
		
		     `name: lastName`
		     `type: string`
		     `validate: max(30)`
		     `mandatory: true`
		
		     `name: age`
		     `type: number`
		     `validate: positive(),min(18)`
		
		     `name: address`
		     `type: address`
		
		Asset: `address`
		
		`name: address`
		
		`type: embedded`
		
		`properties:`
		   `name: street`
		   `type: string`
		
		   `name: city`
		   `type: string`
		
		   `name: state`
		   `type: string`
		
		   `name: country`
		   `type: string`

Table C-2 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`customMethods:`	This property creates invokable custom method templates in the main controller file. It takes the method signature and creates the function declaration in the controller file.
	
	You can provide language specific function declarations here.
	
	We provide a custom method named `executeQuery`. If it's added to the specification file, it details how Berkeley DB SQL and CouchDB rich queries can be executed. This method can be invoked only when you are connected to Oracle Blockchain Platform Cloud or Enterprise Edition.
		TypeScript
		
		`customMethods:`
		`    - executeQuery`
		`    - "buyCar(vin: string, buyerId: string, sellerId: string, price: number, date: Date)"`
		`    - "addCar(vin: string, dealerId: string, price: number, date: Date)"`
		
		Go
		
		`customMethods:`
		`    - executeQuery`
		`    - "BuyCar(vin string, buyerId string, sellerId string, price int)"`
		`    - "AddCar(vin string, dealerId string, price int)"`

Scaffolded TypeScript Chaincode Project

Blockchain App Builder takes the input from your specification file and generates a fully-functional scaffolded chaincode project.

If the chaincode project uses the TypeScript language, the scaffolded project contains three main files:

- `main.ts`
- `<chaincodeName>.model.ts`
- `<chaincodeName>.controller.ts`

All the necessary libraries are installed and packaged. The `tsconfig.json` file contains the necessary configuration to compile and build the TypeScript project.

The `<chaincodeName>.model.ts` contains multiple asset definitions and `<chaincodeName>.controller.ts` contains the assets behavior and CRUD methods.

The various decorators in `model.ts` and `controller.ts` provide support for features like automatic validation of arguments, marshalling/unmarshalling of arguments, transparent persistence capability (ORM) and calling rich queries.

Reference:

- Asset
- Decorators
- Models
- Controller
- CRUD Methods
- Model Methods
- Controller Method Details
- Custom Methods
- Init Method

Asset

By default every class which extends `OchainModel` will have an additional read-only property called `assetType`. This property can be used to fetch only assets of this type. Any changes to this property are ignored during the creation and updating of the asset. The property value by default is `<chaincodeName>.<assetName>`.

```
@Id('supplierId')
export class Supplier extends OchainModel<Supplier> {
    public readonly assetType = 'tsdeml36.supplier';
    @Mandatory()
    @Validate(yup.string())
    public supplierId: string;
```

Decorators

Class decorators
`@Id(identifier)`

This decorator identifies the property which uniquely defines the underlying asset. This property is used as a key of the record, which represents this asset in the chaincode's state. This decorator is automatically applied when a new TypeScript project is scaffolded. The 'identifier' argument of the decorator takes the value from specification file.

```
@Id('supplierId')
export class Supplier extends OchainModel{
...
}
```

Property decorators
Multiple property decorators can be used. The decorators are resolved in top to bottom order.

`@Mandatory()`

This marks the following property as mandatory so it cannot be skipped while saving to the ledger. If skipped it throws an error.

```
@Mandatory()
public supplierID: string;
```

`@Default(param)`

This property can have a default value. The default value in the argument (`param`) is used when the property is skipped while saving to the ledger.

```
@Default('open for business')
@Validate(yup.string())
public remarks: string;
```

`@Validate(param)`

The following property is validated against the schema presented in the parameter. The argument `param` takes a yup schema and many schema methods can be chained together. Many complex validations can be added. Refer to https://www.npmjs.com/package/yup for more details.

```
@Validate(yup.number().min(3))
public productsShipped: number;
```

`@Embedded(PropertyClass)`

This property decorator marks the underlying property as an embeddable asset. It takes the embeddable class as a parameter. This class should extend the `EmbeddedModel` class. This is validated by the decorator.

In this example, `Employee` has a property called `address` of type `Address`, which is to be embedded with the `Employee` asset. This is denoted by the `@Embedded()` decorator.

```
export class Employee extends OchainModel<Employee> {

   public readonly assetType = 'TsSample.employee';

   @Mandatory()
   @Validate(yup.string())
   public emplyeeID: string;

   @Mandatory()
   @Validate(yup.string().max(30))
   public firstName: string;

   @Mandatory()
   @Validate(yup.string().max(30))
   public lastName: string;

   @Validate(yup.number().positive().min(18))
   public age: number;

   @Embedded(Address)
   public address: Address;
}


export class Address extends EmbeddedModel<Address> {

   @Validate(yup.string())
   public street: string;

   @Validate(yup.string())
   public city: string;

   @Validate(yup.string())
   public state: string;

   @Validate(yup.string())
   public country: string;
}
```

When a new instance of the `Address` class is created, all the properties of the `Address` class are automatically validated by the `@Validate()` decorator. Note that the `Address` class does not have the `assetType` property or `@Id()` class decorator. This asset and its properties are not saved in the ledger separately but are saved along with the `Employee` asset. Embedded assets are user defined classes that function as value types. The instance of this class can only be stored in the ledger as a part of the containing object (`OchainModel` assets). All the above decorators are applied automatically based on the input file while scaffolding the project.

`@Derived(STRATEGY, ALGORITHM, FORMAT)`

This decorator is used for defining the attribute derived from other properties. This decorator has two mandatory parameters:

- `STRATEGY`: takes values of `CONCAT` or `HASH`. Requires an additional parameter `ALGORITHM` if `HASH` is selected. The default algorithm is `sha256`; `md5` is also supported.
- `FORMAT`: takes an array of specification strings and values to be used by the strategy.

```
@Id('supplierID')
export class Supplier extends OchainModel<Supplier> {

   public readonly assetType = 'chaincodeTS.supplier';

   @Mandatory()
   @Derived(STRATEGY.HASH.'sha256',['IND%1IND%2','license','name'])
   @Validate(yup.string())
   public supplierID: string;
```

```
    @Validate(yup.string().min(2).max(4))
    public license: string;

    @Validate(yup.string().min(2).max(4))
    public name: string;
```

Method decorators
`@Validator(…params)`

This decorator is applied on methods of the main controller class. This decorator is important for parsing the arguments, validating against all the property decorators and returning a model/type object. It takes multiple user created models or yup schemas as parameter.

Note the order of the parameters should be exactly the same as the order of the arguments in the method.

In this example, the `Supplier` model reference is passed in parameters which corresponds to the `asset` type in the method argument. The decorator in run-time would parse and convert the method argument to JSON object, validate against the `Supplier` validators, and upon successful validation convert the JSON object to `Supplier` object and assign it to the `asset` variable. On completion the underlying method is then finally called.

```
@Validator(Supplier)
public async createSupplier(asset: Supplier) {
    return await asset.save();
}
```

In this example, multiple asset references are passed; they corresponds to the object types of the method arguments. Notice the order in the parameters.

```
@Validator(Supplier, Manufacturer)
public async createProducts(supplier: Supplier, manufacturer: 
Manufacturer) {
}
```

Apart from asset reference, yup schema objects could also be passed if the arguments are of basic-types. In this example, `supplierId` and `rawMaterialSupply` are of type `string` and `number` respectively, so the yup schema of similar type and correct order is passed to the decorator. Notice the chaining of yup schema methods.

```
@Validator(yup.string(), yup.number().positive())
public async fetchRawMaterial(supplierID:string, rawMaterialSupply: 
number) {
    const supplier = await Supplier.get(supplierID);
    supplier.rawMaterialAvailable = supplier.rawMaterialAvailable + 
rawMaterialSupply;
    return await supplier.update();
}
```

Models

Every model class extends `OchainModel`. Transparent Persistence Capability or simplified ORM is captured in the `OchainModel` class. If your model needs to need call any of the below ORM methods, you should extend the `OchainModel` class.

ORM methods which are exposed via `OchainModel`:

- `save` – this calls the Hyperledger Fabric `putState` method
- `get` – this calls the Hyperledger Fabric `getState` method
- `update` – this calls the Hyperledger Fabric `putState` method
- `delete` – this calls the Hyperledger Fabric `deleteState` method
- `history` – this calls the Hyperledger Fabric `getHistoryForKey` method
- `getByRange` – this calls the Hyperledger Fabric `getStateByRange` method

See: Model Methods.

Controller

Main controller class extends `OchainController`. There is only one main controller.

```
export class TSProjectController extends OchainController{
```

You can create any number of classes, functions, or files, but only those methods that are defined within the main controller class are invokable from outside, the rest of them are hidden.

CRUD Methods

As described in Input Specification File, you can specify which CRUD methods you want generated in the specification file. For example, if you selected to generate all methods, the result would be similar to:

```
@Validator(Supplier)
public asynch createSupplier(asset: Supplier){
    return await asset.save();
}
public asynch getSupplierById(id: string){
    const asset = await Supplier.get(id);
    return asset;
}
@Validator(Supplier)
public asynch updateSupplier(asset: Supplier){
    return await asset.update();
}
public asynch deleteSupplier(id: string){
    const result = await Supplier.delete(id);
    return result;
}
public asynch getSupplierHistoryById(id: string){
    const result = await Supplier.history(id);
    return result;
}
```

```
@Validator(yup.string(), yup.string())
public asynch getSupplierByRange(startId: string, endId: string){
    const result = await Supplier.getByRange(startId, endId);
    return result;
}
```

Model Methods

save
The `save` method adds the caller `asset` details to the ledger.

This method calls the Hyperledger Fabric `putState` internally. All marshalling/ unmarshalling is handled internally.

```
<Asset>.save(extraMetadata?: any): Promise<any>
```

Parameters:

- `extraMetadata : any` (optional) – To save metadata apart from the asset into the ledger.

Returns:

- `Promise<any>` - Returns a promise on completion

Example:

```
@Validator(Supplier)
public async createSupplier(asset: Supplier) {
    return await asset.save();
}
```

get
The `get` method is a static method of `OchainModel` class which is inherited by the concrete model classes of `{chaincodeName}.model.ts`.

This returns an asset of `<Asset>` if `id` is found in the ledger and has the same type as `<Asset>`. This method calls the Hyperledger Fabric `getState` method internally. Even though any asset with given`id` is returned from the ledger, our method will take care of casting into the caller `Model` type.

If you would like to return any asset by the given `id`, use the generic controller method `getAssetById`.

```
<Asset>.get(id: string): Promise<asset>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise: <Asset>` - Returns object of type `<Asset>`. Even though any asset with given `id` is returned from the ledger, this method will take care of casting into the caller `Asset` type. If the asset returned from the ledger is not of the `Asset` type, then it throws an error. This check is done by the read-only `assetType` property in the `Model` class.

Example:

```
public async getSupplierById: string) {
    const asset = await Supplier.get(id);
    return asset;
}
```

In the example, `asset` is of the type `Supplier`.

`update`
The `update` method updates the caller `asset` details in the ledger. This method returns a promise.

This method calls the Hyperledger Fabric `putState` internally. All the marshalling/ unmarshalling is handled internally.

```
<Asset>.update(extraMetadata?: any): Promise<any>
```

Parameters:

- `extraMetadata : any` (optional) – To save metadata apart from the asset into the ledger.

Returns:

- `Promise<any>` - Returns a promise on completion

Example:

```
@Validator(Supplier)
public async updateSupplier(asset: Supplier) {
    return await asset.update();
}
```

`delete`
This deletes the asset from the ledger given by `id` if it exists. This method calls the Hyperledger Fabric `deleteState` method internally.

The `delete` method is a static method of `OchainModel` class which is inherited by the concrete `Model` classes of `{chaincodeName}.model.ts`.

```
<Asset>. delete(id: string): Promise<any>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise <any>` - Returns a promise on completion.

Example:

```
public async deleteSupplier(id: string) {
    const result = await Supplier.delete(id);
    return result;
}
```

`history`
The history method is a static method of `OchainModel` class which is inherited by the concrete `Model` classes of `{chaincodeName}.model.ts`. This returns the asset history given by `id` from the ledger, if it exists.

This method calls the Hyperledger Fabric `getHistoryForKey` method internally.

```
<Asset>.history(id: string): Promise<any[]>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise <any[]>` - Returns any [] on completion.

Example

```
public async getSupplierHistoryById(id: string) {
    const result = await Supplier.history(id);
    return result;
}
```

Example of the returned asset history for `getSupplierHistoryById`:

```
[
    {
        "trxId":
"8ef4eae6389e9d592a475c47d7d9fe6253618ca3ae0bcf77b5de57be6d6c3829",
        "timeStamp": 1602568005,
        "isDelete": false,
        "value": {
            "assetType": "supp.supplier",
            "supplierId": "s01",
            "rawMaterialAvailable": 10,
            "license": "abcdabcdabcd",
            "expiryDate": "2020-05-28T18:30:00.000Z",
            "active": true
        }
    },
    {
        "trxId":
"92c772ce41ab75aec2c05d17d7ca9238ce85c33795308296eabfd41ad34e1499",
        "timeStamp": 1602568147,
        "isDelete": false,
        "value": {
            "assetType": "supp.supplier",
            "supplierId": "s01",
            "rawMaterialAvailable": 15,
            "license": "valid license",
            "expiryDate": "2020-05-28T18:30:00.000Z",
            "active": true
        }
    }
]
```

`getByRange`
The `getByRange` method is a static method of `OchainModel` class which is inherited by the concrete `Model` classes of `{chaincodeName}.model.ts`.

This returns a list of asset between the range `startId` and `endId`. This method calls the Hyperledger Fabric `getStateByRange` method internally.

Even though any asset with given `id` is returned from the ledger, our method will take care of casting into the caller `Model` type. In above example, `result` array is of the type `Supplier`. If the asset returned from ledger is not of the `Model` type, then it will not be included in the list. This check is done by the read-only `assetType` property in the `Model` class.

If you would like to return all the assets between the range `startId` and `endId`, use the generic controller method `getAssetsByRange`.

```
<Asset>.getByRange(startId: string, endId: string): Promise<Asset[]>
```

Parameters:

- `startId : string` – Starting key of the range. Included in the range.
- `endId : string` – Ending key of the range. Excluded of the range.

Returns:

- `Promise< Asset[ ] >` - Returns array of `<Asset>` on completion.

Example:

```
@Validator(yup.string(), yup.string())
public async getSupplierByRange(startId: string, endId: string){
    const result = await Supplier.getByRange(startId, endId);
    return result;
}
```

`getId`
When the asset has a derived key as `Id`, you can use this method to get a derived ID. This method will return an error if the derived key contains `%t` (timestamp).

Parameters:

- `object` – Object should contain all the properties on which the derived key is dependent.

Returns:

- Returns the derived key as a string.

Example:

```
@Validator(yup.string(), yup.string())
public async customGetterForSupplier(license: string, name: string){
    let object = {
      license : license,
      name: name
    }
    const id = await Supplier.getID(object);
    return Supplier.get(id);
}
```

Controller Method Details

Apart from the above model CRUD and non-CRUD methods, Blockchain App Builder provides out-of-the box support for other Hyperledger Fabric methods from our controller. These methods are:

- `getAssetById`
- `getAssetsByRange`
- `getAssetHistoryById`
- `query`
- `generateCompositeKey`
- `getByCompositeKey`
- `getTransactionId`
- `getTransactionTimestamp`
- `getTransactionInvoker`
- `getChannelID`
- `getCreator`
- `getSignedProposal`
- `getArgs`
- `getStringArgs`
- `getMspID`
- `getNetworkStub`

These methods are available with `this` context itself in the `Controller` class. For example:

```
public async getModelById(id: string) {
    const asset = await this.getAssetById(id);
    return asset;
}
@Validator(yup.string(), yup.string())
public async getModelsByRange(startId: string, endId: string) {
    const asset = await this.getAssetsByRange(startId, endId);
    return asset;
}
public async getModelHistoryById(id: string) {
    const result = await this.getAssetHistoryById(id);
    return result;
}
```

`getAssetById`
The `getAssetById` method returns asset based on `id` provided. This is a generic method and be used to get asset of any type.

```
this< OchainController>.getAssetById(id: string): Promise<byte[]>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise <byte [ ]>` - Returns promise on completion. You have to convert `byte[]` into an object.

`getAssetsByRange`
The `getAssetsByRange` method returns all assets present from `startId` (inclusive) to `endId` (exclusive) irrespective of asset types. This is a generic method and can be used to get assets of any type.

```
this<OchainController>.getAssetsByRange(startId: string, endId:
string):
Promise<shim.Iterators.StateQueryIterator>
```

Parameters:

- `startId : string` – Starting key of the range. Included in the range.
- `endId : string` – Ending key of the range. Excluded of the range.

Returns:

- `Promise< shim.Iterators.StateQueryIterator>` - Returns an iterator on completion. You have to iterate over it.

`getAssetHistoryById`
The `getAssetHistoryById` method returns history iterator of an asset for `id` provided.

```
this<OchainController>.getAssetHistoryById(id: string):
Promise<shim.Iterators.HistoryQueryIterator>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise<shim.Iterators.HistoryQueryIterator>` - Returns a history query iterator. You have to iterate over it.

`query`
The `query` method will run a Rich SQL/Couch DB query over the ledger. This method is only supported for remote deployment on Oracle Blockchain Platform. This is a generic method for executing SQL queries on the ledger.

```
this<OchainController>.query(queryStr: string):
Promise<shim.Iterators.StateQueryIterator>
```

Parameters:

- `queryStr : string` - Rich SQL/Couch DB query.

Returns:

- `Promise<shim.Iterators.StateQueryIterator>` - Returns a state query iterator. You have to iterate over it.

`generateCompositeKey`
This method generates and returns the composite key based on the `indexName` and the attributes given in the arguments.

```
this<OchainController>.generateCompositeKey(indexName: string,
attributes:
string[]): string
```

Parameters:

- `indexName : string` - Object Type of the key used to save data into the ledger.
- `attributes: string[ ]` - Attributes based on which composite key will be formed.

Returns:

- `string` - Returns a composite key.

`getByCompositeKey`
This method returns the asset that matches the key and the column given in the attribute parameter while creating composite key. `indexOfId` parameter indicates the index of the key returned in the array of stub method `SplitCompositeKey`.

Internally this method calls Hyperledger Fabric's `getStateByPartialCompositeKey`, `splitCompositeKey` and `getState`.

```
this<OchainController>.getByCompositeKey(key: string, columns:
string[],
indexOfId: number): Promise<any []>
```

Parameters:

- `key: string` – Key used to save data into ledger.
- `columns: string[ ]` - Attributes based on key is generated.
- `indexOfId: number` - Index of attribute to be retrieved from Key.

Returns:

- `Promise< any [ ]` - Returns any [] on completion.

`getTransactionId`
Returns the transaction ID for the current chaincode invocation request. The transaction ID uniquely identifies the transaction within the scope of the channel.

```
this<OchainController>.getTransactionId(): string
```

Parameters:

- none

Returns:

- `string` - Returns the transaction ID for the current chaincode invocation request.

`getTransactionTimestamp`
Returns the timestamp when the transaction was created. This is taken from the transaction `ChannelHeader`, therefore it will indicate the client's timestamp, and will have the same value across all endorsers.

```
this<OchainController>.getTransactionTimestamp(): Timestamp
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Timestamp` - Returns the timestamp when the transaction was created.

`getTransactionInvoker`
Returns the caller of the transaction from the Transient map property `bcsRestClientId`.

```
this<OchainController>.getTransactionInvoker(): string
```

Parameters:

- none

Returns:

- `string` - Returns the caller of the transaction.

`getChannelID`
Returns the channel ID for the proposal for chaincode to process.

```
this<OchainController>.getChannelID(): string
```

Parameters:

- none

Returns:

- `string` - Returns the channel ID.

`getCreator`
Returns the identity object of the chaincode invocation's submitter.

```
this<OchainController>.getCreator(): shim.SerializedIdentity
```

Parameters:

- none

Returns:

- `shim.SerializedIdentity` - Returns identity object.

`getSignedProposal`
Returns a fully decoded object of the signed transaction proposal.

```
this<OchainController>.getSignedProposal():
shim.ChaincodeProposal.SignedProposal
```

Parameters:

- none

Returns:

- `shim.ChaincodeProposal.SignedProposal` - Returns decoded object of the signed transaction proposal.

`getArgs`
Returns the arguments as array of strings from the chaincode invocation request.

```
this<OchainController>.getArgs(): string[]
```

Parameters:

- none

Returns:

- `string [ ]` - Returns arguments as array of strings from the chaincode invocation.

`getStringArgs`
Returns the arguments as array of strings from the chaincode invocation request.

```
this<OchainController>.getStringArgs(): string[]
```

Parameters:

- none

Returns:

- `string [ ]` - Returns arguments as array of strings from the chaincode invocation.

`getMspID`
Returns the MSP ID of the invoking identity.

```
this<OchainController>.getMspID(): string
```

Parameters:

- none

Returns:

- `string` - Returns the MSP ID of the invoking identity.

`getNetworkStub`
The user can get access to the shim stub by calling `getNetworkStub` method. This will help user to write its own implementation of working directly with the assets.

```
this<OchainController>.getNetworkStub(): shim.ChaincodeStub
```

Parameters:

- none

Returns:

- `shim.ChaincodeStub` - Returns chaincode network stub.

Custom Methods

The following custom methods were generated from our example specification file.

The `executeQuery` shows how SQL rich queries can be called. The validators against the arguments are added automatically by Blockchain App Builder based on the type of the argument specified in the specification file.

```
/**
*
*    BDB sql rich queries can be executed in OBP CS/EE.
*    This method can be invoked only when connected to remote OBP CS/EE
network.
*
*/
@Validator(yup.string()}
public async executeQuery(query: string) {
    const result = await OchainController.query(query);
    return result;
}
@Validator(yup.string(), yup.number()}
public async fetchRawMaterial(supplierId: string, rawMaterialSupply:
number) {
}
@Validator(yup.string(), yup.string(), yup.number())
public async getRawMaterialFromSupplier(manufacturerId: string,
supplierId: string, rawMaterialSupply: number) {
}
@Validator(yup.string(), yup.number(), yup.number())
public async createProducts(manufacturerId: string,
rawMaterialConsumed: number, productsCreated: number) {
}
public async sendProductsToDistribution() {
}
```

Init Method

We have provided one `init` method in the controller with an empty definition. This method will be called by the Hyperledger Fabric `Init` method during first time instantiating or upgrading the chaincode.

```
export class TestTsProjectController extends OchainController {
    public async init(params: any) {
        return;
}
```

If you would like to initialize any application state at this point, you can use this method to do that.

Scaffolded Go Chaincode Project

Blockchain App Builder takes the input from your specification file and generates a fully-functional scaffolded chaincode project.

If the chaincode project is in the Go language, the scaffolded project contains three main files:

- `main.go`
- `<chaincodeName>.model.go`
- `<chaincodeName>.controller.go`

All the necessary libraries are installed and packaged.

The `<chaincodeName>.model.go` contains multiple asset definitions and `<chaincodeName>.controller.go` contains the asset's behavior and CRUD methods. The various Go struct tags and packages in `model.go` and `controller.go` provide support for features like automatic validation of arguments, marshalling/unmarshalling of arguments, transparent persistence capability (ORM) and calling rich queries.

The scaffolded project can be found in `$GOPATH/src/example.com/<chaincodeName>`

Reference:

- Validators
- Model
- Composite Key Methods
- Stub Method
- Other Methods
- Utility Package
- Controller
- CRUD Methods
- Custom Methods
- Init Method

Validators

Id
`id:"true"`

This validator identifies the property which uniquely defines the underlying model. The asset is saved by the value in this key. This validator automatically applies when a new Go project is scaffolded.

In the below screenshot `"SupplierId"` is the key for the supplier asset and has a tag property `id:"true"` for the SupplierId property.

```
type Supplier struct {
    Supplierld              string        'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable    int           'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License                 string        'json:"License"
validate:"string,min=10"'
    ExpiryDate              date.Date     'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                   bool          'json:"Active"
validate:"bool" default :"true"'
    Metadata                 interface{}
'json:"Metadata,omitempty"'
}
```

Derived
`derived:"strategy,algorithm,format"`

This decorator is used for defining the attribute derived from other properties. This decorator has two mandatory parameters:

- `strategy`: takes values of `concat` or `hash`. Requires an additional parameter `algorithm` if `hash` is selected. The default algorithm is `sha256`; `md5` is also supported.
- `format`: takes an array of specification strings and values to be used by the strategy.

```
type Supplier struct{
   AssetType string 'json:"AssetType" final:"chaincode1.Supplier"'
   SupplierId   string  'json:"SupplierId"
validate:"string" id:"true" mandatory:"true"
derived:"strategy=hash,algorith=sha256,format=IND%1%2,License,Name"'
   Name         string  'json:"Name" validate:"string,min=2,max=4"'
   License      string  'json:"License" validate:"string,min=2,max=4"'
}
```

Mandatory
`validate:"mandatory"`

This marks the following property as mandatory and cannot be skipped while saving to the ledger. If skipped it throws an error. In the below example, "`SupplierId`" has a `validate:"mandatory"` tag.

```
Type Supplier struct {
    Supplierld               string      'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable     int         'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License                  string      'json:"License"
validate:"string,min=10"'
    ExpiryDate               date.Date   'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                   bool        'json:"Active" validate:"bool"
default :"true"'
    Metadata                 interface{} 'json:"Metadata,omitempty"'
}
```

Default
`default:"<param>"`

This states that the following property can have a default value. The default value in the default tag is used when the property is skipped while saving to the ledger. In the below example property, `Active` has a default value of `true`, provided as tag `default:"true"`

```
Type Supplier struct {
    Supplierld             string       'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License                string       'json:"License"
validate:"string,min=10"'
    ExpiryDate             date.Date    'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                 bool         'json:"Active" validate:"bool"
default :"true"'
    Metadata               interface{}  'json:"Metadata,omitempty"'
}
```

Validate types
Basic Go types are validated for a property by defining a validate tag. These are the validate tags based on types:

- string: `validate: "string"`
- date: `validate: "date"`
- number: `validate: "int"`
- boolean: `validate: "bool"`

Min validator
`validate:"min=<param>"`

Using the min validator, minimum value can be set for a property of type number and string.

For type int: In the example, `RawMaterialAvailable` property has a minimum value of 0 and if a value less than 0 is applied to `RawMaterialAvailable` an error will be returned.

For type string: For the string type minimum validator will check the length of the string with the provided value. Therefore, in the below example the `License` property has to be minimum 10 characters long.

Example:

```
Type Supplier struct {
    Supplierld             string       'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable   int          'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License                string       'json:"License"
validate:"string,min=10"'
    ExpiryDate             date.Date    'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                  bool         'json:"Active" validate:"bool"
default :"true"'
    Metadata                interface{}  'json:"Metadata,omitempty"'
}
```

Max validator
`validate:"max=<param>"`

Using the max validator, the maximum value can be set for a property of type number and string.

For type int: Like the min validator, for type int, if a value provided for the `structfield` is greater than the value provided in the validator then an error will be returned.

For type string: Like the min validator, max validator will also check the length of the string with given value. In the example, the `Domian` property has a maximum value of 50, so if the `Domain` property has a string length more than 50 characters, then an error message will be returned.

```
type Retailer struct {
    Retailerld         string        'json:"Retailerld"
validate:"string,mandatory" id:"true"'
    ProductsOrdered    int           'json:"ProductsOrdered"
validate:"int,mandatory"'
    ProductsAvailable  int           'json:"ProductsAvailable"
validate:"int" default:"1"'
    ProductsSold       int           'json:"ProductsSold"
validate:"int"'
    Remarks            string        'json:"Remarks" validate:"string"
default :"open for business"'
    Items              []int         'json:"Items"
validate:"array=int,range=l-5"'
    Domain             string        'json:"Domain"
validate:"url,min=30,max=50"'
    Metadata           interface{}   'json:"Metadata,omitempty"'
}
```

Date validators
Before validator:

`validate:"before=<param>"`

The before validator validates a property of type `date` to have a value less than the specified in parameter.

In this example, the `ExpiryDate` property should be before `"2020-06-26"` and if not it will return an error.

```
Type Supplier struct {
    Supplierld            string       'json:"Supplierld"
validate:"string,mandatory" id:"true"'
```

```
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License               string       'json:"License"
validate:"string,min=10"'
    ExpiryDate            date.Date    'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                bool         'json:"Active" validate:"bool"
default :"true"'
    Metadata              interface{}  'json:"Metadata,omitempty"'
}
```

After validator:

```
validate:"after=<param>"
```

The before validator validates a property of type `date` to have a value greater than the specified in parameter.

In this example, the `CompletionDate` property should be after "`2020-06-26`" and if not it will return an error.

```
Type Supplier struct {
    Manufacturerld        string       'json:"Manufacturerld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,max=8"'
    ProductsAvailable     int          'json:"ProductsAvailable"
validate:"int"'
    CompletionDate        date.Date    'json:"CompletionDate"
validate:"date,after=2020-06-26"'
    Metadata              interface{}  'json:"Metadata,omitempty"'
}
```

URL validator
`validate:"url"`

The URL validator will validate a property for URL strings.

In this example, the `Domain` property has to be a valid URL.

```
type Retailer struct {
    Retailerld         string        'json:"Retailerld"
validate:"string,mandatory" id:"true"'
    ProductsOrdered    int           'json:"ProductsOrdered"
validate:"int,mandatory"'
    ProductsAvailable  int           'json:"ProductsAvailable"
validate:"int" default:"1"'
    ProductsSold       int           'json:"ProductsSold"
validate:"int"'
    Remarks            string        'json:"Remarks" validate:"string"
default :"open for business"'
    Items              []int         'json:"Items"
validate:"array=int,range=l-5"'
    Domain             string        'json:"Domain"
validate:"string,url,min=30,max=50"'
    Metadata           interface{}   'json:"Metadata,omitempty"'
}
```

Regexp validator

`validate:"regexp=<param>"`

Regexp validator will validate property for the input regular expression.

In this example, the `PhoneNumber` property will validate for a mobile number as per the regular expression.

```
type Customer struct {
Customerld      string       'json:"Customerld"
validate:"string,mandatory" id:"true"'
Name            string       'json:"Name" validate:"string,mandatory"'
ProductsBought  int          'json:"ProductsBought" validate:"int"'
OfferApplied    int          'json:"OfferApplied"
validate :"int,nax=0"'
PhoneNumber     string
'json:"PhoneNumber" validate:"string,regexp=A\(?([0-9]{3})\)?[-. ]?
([0-9]{3})[-. ]?([0-9]{4})$"'
Received        bool         'json:"Received" validate:"bool“'
Metadata        interface{}  'json:"Metadata,omitempty"'
}
```

Multiple validators

Multiple validators can be applied a property.

In this example, the `Domain` property has validation for a string, URL, and min and max string length.

```
type Retailer struct {
    Retailerld         string        'json:"Retailerld"
validate:"string,mandatory" id:"true"'
    ProductsOrdered    int           'json:"ProductsOrdered"
validate:"int,mandatory"'
    ProductsAvailable  int           'json:"ProductsAvailable"
validate:"int" default:"1"'
    ProductsSold       int           'json:"ProductsSold"
validate:"int"'
    Remarks            string        'json:"Remarks" validate:"string"
default :"open for business"'
    Items              []int         'json:"Items"
validate:"array=int,range=l-5"'
    Domain             string        'json:"Domain"
validate:"string,url,min=30,max=50"'
    Metadata           interface{}   'json:"Metadata,omitempty"'
}
```

Model

Asset Type Property

By default every struct will have an additional property called `AssetType`. This property can be useful in fetching only assets of this type. Any changes to this property is ignored during create and update of asset. The property value by default is `<chaincodeName>.<modelName>`.

```
type Supplier struct {
AssetType string 'json:"AssetType" default:"TestGoProject.Supplier"'

SupplierId            string      'json:"SupplierId"
validate:"string,mandatory" id:"true'
RawMaterialAvailable  int         'json:"RawMaterialAvailable"
validate:"int,min=0"'
License               string      'json:"License"
validate:"string,min=10"'
ExpiryDate            date.Date   'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
Active                bool      'json:"Active" validate:"bool"
default:"true"'
Metadata              interface{} 'json:"Metadata,omitempty"'
}
```

ORM

Go chaincodes implement Transparent Persistence Capability (ORM) with the model package.

The following ORM methods are exposed via the model package:

`model.Get`
Queries the ledger for the stored asset based on the given ID.

```
func Get(Id string, result ...interface{}) (interface{}, error)
```

Parameters:

- `Id` - The ID of the asset which is required from the ledger.
- `result (interface{})` - This is an empty asset object of a particular type, which is passed by reference. This object will contain the result from this method. To be used only if type-specific result is required.
- `asset (interface)` - Empty asset object, which is passed by reference. This object will contain the result from this method. To be used only if type-specific result is required.

Returns:

- `interface {}` - Interface contains the asset in the form of `map[string]interface{}`. Before operating on this map, it is required to assert the obtained interface with type `map[string]interface{}`. To convert this map into an asset object, you can use the utility API `util.ConvertMaptoStruct` (see: Utility Package).

- `error` - Contains an error if returned, or is nil.

`model.Update`
Updates the provided asset in the ledger with the new values.

```
func Update(args ...interface{}) (interface{}, error)
```

Parameters:

- `obj (interface)` - The object that is required to be updated in the ledger is passed by reference into this API with the new values. The input asset is validated and verified according to the struct tags mentioned in the model specification and then stored into the ledger.

Returns:

- `interface{}` - The saved asset is returned as an interface.
- `error` - Contains an error if returned, or is nil.

`model.Save`
Saves the asset to the ledger after validating on all the struct tags.

```
func Save(args ...interface{}) (interface{}, error)
```

Parameters:

- `obj/args[0] (interface{})` - The object that needs to be stored in the ledger is passed by reference in this utility method.
- `metadata/args[1] (interface{})` - This parameter is optional. It has been given in order to facilitate you if you're required to store any metadata into the ledger along with the asset at the runtime. This parameter can be skipped if no such requirement exists.

Returns:

- `interface {}` - The asset is returned as an interface.
- `error` - Contains an error if returned, or is nil.

`model.Delete`
Deletes the asset from the ledger.

```
func Delete(Id string) (interface{}, error)
```

Parameters:

- `id (string)` - The ID of the asset which is required to be deleted from the ledger.

Returns:

- `interface {}` - Contains the asset being deleted in the form of `map[string]interface{}`.

`model.GetByRange`
Returns the list of assets by range of IDs.

```
func GetByRange(startKey string, endKey string, asset ...interface{})
([]map[string]interface{}, error)
```

Parameters:

- `startkey (string)` - Starting ID for the range of objects which are required.
- `endkey (string)` - End of the range of objects which are required.
- `asset interface` - (optional) Empty array of assets, which is passed by reference. This array will contain the result from this method. To be used if type-specific result is required.

Returns:

- `[]map[string]interface{}` - This array contains the list of assets obtained from the ledger. You can access the objects iterating over this array and asserting the objects as `map[string]interface{}` and using utility to convert to asset object.
- `error` - Contains an error if returned, or is nil.

`model.GetHistoryById`
Returns the history of the asset with the given ID.

```
func GetHistoryByID(Id string) ([]interface{}, error)
```

Parameters:

- `Id (string)` - ID of the asset for which the history is needed.

Returns:

- `[]interface{}` - This slice contains the history of the asset obtained from the ledger in form of slice of `map[string]interface{}`. You can access each history element by iterating over this slice and asserting the objects as `map[string]interface{}` and using utility to convert to asset object.
- `error` - Contains the error if observed.

`model.Query`
The query method will run a SQL/Couch DB query over the ledger. This method is only supported for remote deployment on Oracle Blockchain Platform. This is a generic method for executing SQL queries on the ledger.

```
func Query(queryString string) ([]interface{}, error)
```

Parameters:

- `queryString (string)` - Input the query string.

Returns:

- `[]interface{}` - This will contain the output of the query. The result is in form of slice of interfaces. You need to iterate over the slice and use the elements by converting them to proper types.
- `error` - Contains the error if observed.

Composite Key Methods

`model.GenerateCompositeKey`
This method generates and returns the composite key based on the `indexName` and the attributes given in the arguments.

```
func GenerateCompositeKey(indexName string, attributes []string)
(string, error)
```

Parameters:

- `indexName (string)` - Object type of the composite key.
- `attrbutes ([]string)` - Attributes of the asset based on which the composite key has to be formed.

Returns:

- `string` - This contains the composite key result.
- `error` - Contains the error if observed.

`model.GetByCompositeKey`
This method returns the asset that matches the key and the column given in the parameters. The `index` parameter indicates the index of the key returned in the array of stub method `SplitCompositeKey`.

Internally this method calls Hyperledger Fabric's `getStateByPartialCompositeKey`, `splitCompositeKey` and `getState`.

```
func GetByCompositeKey(key string, columns []string, index int)
(interface{}, error)
```

Parameters:

- `key (string)` - Object type provided while creating composite key.
- `column ([]string)` - This is the slice of attributes on which the ledger has to be queried using the composite key.
- `index(int)` - Index of the attribute.

Returns:

- `Interface{}` - Contains the list of assets which are result of this method.
- `error` - Contains any errors if present.

Stub Method

`model.GetNetworkStub`
This method will return the Hyperledger Fabric `chaincodeStub`.

You can get access to the shim stub by calling the `GetNetworkStub` method. This will help you write your own implementation working directly with the assets.

```
func GetNetworkStub() shim.ChaincodeStubInterface
```

Parameters:

- none

Returns:

- `shim.ChaincodeStubInterface` - This is the Hyperledger Fabric chaincode stub.

Other Methods

- `model.GetTransactionId()`
- `model.GetTransactionTimestamp()`
- `model.GetChannelID()`
- `model.GetCreator()`
- `model.GetSignedProposal()`
- `model.GetArgs()`
- `model.GetStringArgs()`
- `model.getId`

`model.GetTransactionId`
Returns the transaction ID for the current chaincode invocation request. The transaction ID uniquely identifies the transaction within the scope of the channel.

```
func GetTransactionId() string
```

Parameters:

- none

Returns:

- `string` - This contains the required transaction ID.

`model.GetTransactionTimestamp`
Returns the timestamp when the transaction was created. This is taken from the transaction `ChannelHeader`, therefore it will indicate the client's timestamp, and will have the same value across all endorsers.

```
func GetTransactionTimestamp() (*timestamp.Timestamp, error)
```

Parameters:

- none

Returns:

- `timestamp.Timestamp` - Contains the timestamp required.
- `error` - Contains any errors if present.

`model.GetChannelID`
Returns the channel ID for the proposal for the chaincode to process.

```
func GetChannelID() string
```

Parameters:

- none

Returns:

- `string` - Contains the required channel ID as a string.

`model.GetCreator`
Returns the identity object of the chaincode invocation's submitter

```
func GetCreator() ([]byte, error)
```

Parameters:

- none

Returns:

- `[ ]byte` - Contains the required identity object serialized.
- `error` - Contains any errors if present.

`model.GetSignedProposal`
Returns a fully decoded object of the signed transaction proposal.

```
func GetSignedProposal() (*peer.SignedProposal, error)
```

Parameters:

- none

Returns:

- `*peer.SignedProposal` - Contains the required signed proposal object.
- `error` - Contains any errors if present.

`model.GetArgs`
Returns the arguments as array of strings from the chaincode invocation request.

```
func GetArgs() [][]byte
```

Parameters:

- none

Returns:

- `[ ][ ]byte` - Contains the arguments passed.

`model.GetStringArgs`
Returns the arguments intended for the chaincode Init and Invoke as a string array.

```
func GetStringArgs() []string
```

Parameters:

- none

Returns:

- `[]string` - Contains the required arguments as a string array.

`model.getId`
When the asset has a derived key as `Id`, you can use this method to get a derived ID. This method will return an error if the derived key contains `%t` (timestamp).

Parameters:

- `object` - Object should contain all the properties on which the derived key is dependent.

Returns:

- Returns the derived key as a string.

Example:

```
func (t *Controller) CustomGetterForSupplier(License string, Name
string)(interface{}, error){
   var asset Supplier
   asset.License = License
   asset.Name = Name
   id,err := model.GetId(&asset)
   if err !=nil {
      return nil, fmt.Errorf("error in getting ID %s", err.Error())
   }
   return t.GetSupplierById(id)
}
```

Utility Package

The following methods in the utility package may be useful:

`Util.CreateModel`
Parses the provided JSON string and creates an asset object of the provided type.

```
func CreateModel(obj interface{}, inputString string) error
```

Parameters:

- `inputString (string)` - The input JSON string from which the object is to be created.
- `obj (interface{})` - The reference of the object that is to be created from the JSON string. This object will store the created model which is also validated as per validator tags.

Returns:

- `error` - Contains any errors found while creating or validating the asset.

`util.ConvertMapToStruct`
Convert the provided map into object of provided type.

```
func ConvertMapToStruct(inputMap map[string](interface{}), resultStruct
interface{}) error
```

Parameters:

- `inputMap (map[string](interface{}))` - Map which needs to be converted into the asset object.
- `resultStruct (interface{})` - The reference of the required asset object which needs to be generated from the map. Contains the result asset object required.

Returns:

- `error` - Contains any errors found while creating or validating the asset.

Controller

The `Controller.go` file implements the CRUD and custom methods for the assets.

You can create any number of classes, functions, or files, but only those methods that are defined on chaincode struct are invokable from outside, the rest of them are hidden.

CRUD Methods

As described in Input Specification File, you can specify which CRUD methods you want generated in the specification file. For example, if you selected to generate all methods, the result would be similar to:

```
//
//Supplier
//
func (t *ChainCode) CreateSupplier(inputString string) (interface{},
error) {
    var obj Supplier
    err := util.CreateModel(&obj, inputString)
    if err != nil {
        return nil, err
    }
    return model.Save(&obj)
}

func (t *ChainCode) GetSupplierById(id string) (interface{}, error) {
    asset, err := model.Get(id)
    return asset, err
}

func (t *ChainCode) UpdateSupplier(inputString string) (interface{},
error) {
    var obj Supplier
    err := util.CreateModel(&obj, inputstring)
    if err != nil {
        return nil, err
    }
return model.Update(&obj)
}

func (t *ChainCode) DeleteSupplier(id string) (interface{}, error) {
    return model.Delete(id)
}

func (t *ChainCode) GetSupplierHistoryById(id string) (interface{},
error) {
    historyArray, err := model.GetHistoryByld(id)
    return historyArray, err
}

func (t *ChainCode) GetSupplierByRange(startkey string, endKey string)
(interface{}, error) {
    assetArray, err := model.GetByRange(startkey, endKey)
    return assetArray, err
}
```

Custom Methods

The following custom methods were generated from our example specification file.

The `executeQuery` shows how SQL rich queries can be called. The validators against the arguments are added automatically by Blockchain App Builder based on the type of the argument specified in the specification file.

You can implement the functionality as per the business logic.

```
//
//Custom Methods
//
/*
*    BDB sql rich queries can be executed in OBP CS/EE.
*    This method can be invoked only when connected to remote OBP CS/EE
network.
*/
```

```
func (t *ChainCode) ExecuteQuery(inputQuery string) (interface{},
error) {
    resultArray, err := model.Query(inputQuery)
    return resultArray, err
}

func (t *ChainCode) FetchRawMaterial(supplierId string,
rawMaterialSupply int) (interface{}, error) {
    return nil, nil
}

func (t *ChainCode) GetRawMaterialFromSupplier(manufacturerId string,
supplierId string, rawMaterialSupply int) (interface{} error) {
    return nil, nil
}

Func (t *ChainCode) CreateProducts(manufacturerId string,
rawMaterialConsumed int, productsCreated int) (interface{}, error) {
    return nil, nil
}

func (t *ChainCode) SendProductsToDistribution() (interface{}, error) {
    return nil, nil
}
```

For Go chaincodes, every custom method should return two values: *empty interface*, *error*. For example:

```
func (t *Controller) FetchRawMaterial(supplierId string,
rawMaterialSupply int) (interface{}, error) {
    return nil, nil
}
```

Init Method

We have provided one `init` method in the controller with an empty definition. This method will be called by the Hyperledger Fabric `Init` method during first time instantiation or upgrade of a chaincode.

```
type Controller struct {
}
func (t *Controller) Init(args string) (interface{}, error)
    { return nil, nil
}
```

If you would like to initialize any application state at this point, you can use this method to do that.

Deploy Your Chaincode Using the CLI

Once your chaincode project is created, you can deploy it locally to the automatically generated Hyperledger Fabric network, or remotely to your Oracle Blockchain Platform

Cloud or Enterprise Edition. You can also package the chaincode project for manual deployment to Oracle Blockchain Platform.

- Deploy Your Chaincode to a Local Hyperledger Fabric Network
- Deploy Your Chaincode to a Remote Oracle Blockchain Platform Network
- Package Your Chaincode Project for Manual Deployment to Oracle Blockchain Platform

Deploy Your Chaincode to a Local Hyperledger Fabric Network

Once you have created your chaincode project, you can deploy it to a local Hyperledger Fabric network. This basic single-channel network is created for you when you install Blockchain App Builder.

The Blockchain App Builder `ochain run` command starts the Hyperledger Fabric network, other services, and installs and instantiates the chaincode for you.

```
my-mac:GOProject myname$ ochain run -h
Usage: run [options] [...args]
Run chaincode project locally in debug mode.

Arguments :
[...args] (optional) Chaincode instantiate arguments. Arguments should
be space separated.

Options:
-h, --help        output command usage information
-D, --debug       enable debug logging
-P, --debug-port  (optional) specify debug process port
-b, --build       (optional) rebuild runtime if already exists
-p, --project     (optional) Path to Chaincode project to run. If not
specified, it defaults to current directory.

Examples :
$> ochain run
```

If you would like to see the debug logs, you can pass the `--debug` option to the command. You can run the basic network and deploy the chaincode on a different port from the default by passing the `--port` option to the command.

Verifying

The following logs show that the chaincode has been installed and instantiated successfully.

```
my-mac:TSProject myname$ ochain run
Recreating orderer.example.com ... done
Recreating ca.example.com ... done
Recreating peer0.org1.example.com ... done
[2020-09-23T18: 04:09.132] [INFO] default -
============ Started Install Chaincode ============
[2020-09-23T18:04:09.193] [INFO] default Chaincode TSProject:l not
installed.
[2020-09-23T18:04:09.317] [INFO] default - Successfully sent install
```

```
Proposal and received ProposalResponse
[2020-09-23T18:04:09.317] [INFO] default - Successfully installed
chaincode TSProject
[2020-09-23T18:04:09.317] [INFO] default -
============ Finished Install Chaincode ============
[2020-09-23T18:04:09.317] [INFO] default - Successfully installed
chaincode TSProject
[2020-09-23T18:04:09.318] [INFO] default -
============ started instantiate Chaincode ============
[2020-09-23T18:04:09.366] [INFO] default - Successfully sent Proposal
and received ProposalResponse
[2020-09-23T18:04:11.434] [INFO] default - The chaincode instantiate
transaction has been committed on peer localhost:7051
[2020-09-23T18:04:11.434] [INFO] default - The chaincode instantiate
transaction was valid.
[2020-09-23T18:04:11.435] [INFO] default - Successfully sent
transaction to the orderer.
[2020-09-23T18:04:11.435] [INFO] default - Successfully instantiated
chaincode TSProject on channel mychannel
[2020-09-23T18:04:11.435] [INFO] default -
============ Finished instantiate Chaincode ============
[2020-09-23T18:04:11.4351 INFO] default - Successfully instantiated
chaincode TSProject on channel mychannel
INFO (Runtime): Chaincode TSProject installed and ready:
INFO (RunCommand): Chaincode TSProject deployed
```

Troubleshooting

You may encounter the following issues when running your chaincode project on a local network.

Missing Go permissions
While installing Go chaincode project in local network, you might see an error similar to the following:

```
My-Mac:GoProj myname$ ochain run
Starting ca.example.com    ... done
Starting orderer.example.com ... done
Starting peer0.orgl.example.com ... done
INFO (Runtime): 2020/06/22 22:57:09 build started

INFO (Runtime): Building ....

INFO (Runtime): go build runtime/cgo: copying /Users/myname/
Library/Caches/go-build/f8/…….….d: open /usr/local/go/pkg/darwin_amd64/
runtine/
cgo.a: permission denied

ERROR (Runtime): go build runtine/cgo: copying /Users/myname/Library/
Caches/go-build/f8/.…..….d: open /usr/local/go/pkg/darwin_amd64/runtime/
cgo.a: permission denied
```

```
INFO (Runtime): An error occurred while building: exit status 1

Stopping peer0.orgl.exmple.com ... done
Stopping ca.example.com     ... done
Stopping orderer.example.con ... done
```

This is due to missing permissions for Go. This error has been seen only in Mac OS. This is a known issue:

- https://github.com/golang/go/issues/37962
- https://github.com/golang/go/issues/24674
- https://github.com/udhos/update-golang/issues/15

Solution: change the permissions of your `$GOROOT` and try `ochain run` again:

```
sudo chmod -R 777 /usr/local/go
```

Instantiation failure
Due to instantiation failure, corrupt instantiation, Docker peer container full, or Docker peer was killed in local network, you may see an error similar to:

```
============ Started instantiate Chaincode ============
[2028-19-01T19:25:lO.372] [ERROR] default - Error instantiating
Chaincode GollGl on channel mychannel, detailed
error: Error: error starting container: error starting container:
Failed to generate platform-specific docker
build: Failed to pull hyperledger/fabric-ccenv:latest : API error
(404): manifest for hyperledger/
fabric-ccenv:latest not found: manifest unknown: manifest unknown
[2020-19-01T19:25:10.372] (INFO) default -
============ Finished instantiate Chaincode ============
[2020-19-01119:25:10.372] [ERROR] default - Error: Error instantiating
Chaincode Goll01 on channel mychannel,
detailed error: Error: error starting container: error starting
container: Failed to generate platfom-specific
docker build: Failed to pull hyperledger/fabric-ccenv: latest : API
error (404): manifest for hyperledger/
fabric-ccenv:lalest not found: manifest unknown: manifest unknown
exited: signal: terminated
INFO: exited: signal: terminated

ERROR: Error in Chaincode deployment
```

This is due to a peer container not able to start up properly again.

Solution: try the `ochain run` command again, but with the `-b` option. This option rebuilds the runtime for you.

```
ochain run -b
```

Environment Rebuild Required

You may see an error similar to:

```
Starting ca.example.com ...
Starting orderer.example.com ...
Starting orderer.example.com ... error
ERROR: for orderer.example.com
Cannot start service orderer.example.com:
error while creating mount source
path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/config': mkdir /host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0: operation not permitted
Starting ca.example.com... error
ERROR: for ca.example.com
Cannot start service ca.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/crypto-config/peerOrganizations/org1.example.com/
ca': mkdir /host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0: operation not permitted
ERROR: for orderer.example.com
Cannot start service orderer.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/config': mkdir /host_mnt/c/Users/opc/.vscode/
extensions/oracle.oracle-blockchain-1.4.0: operation not permitted
ERROR: for ca.example.com
Cannot start service ca.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/crypto-config/peerOrganizations/org1.example.com/
ca': mkdir /host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0: operation not permitted
Encountered errors while bringing up the project.
ERROR: Starting ca.example.com ...
Starting orderer.example.com ...
Starting orderer.example.com ... error
ERROR: for orderer.example.com
Cannot start service orderer.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/config': mkdir /host_mnt/c/Users/opc/.vscode/
extensions/oracle.oracle-blockchain-1.4.0: operation not permitted
Starting ca.example.com ... error
ERROR: for ca.example.com
Cannot start service ca.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/crypto-config/peerOrganizations/org1.example.com/
ca': mkdir /host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0: operation not permitted
ERROR: for orderer.example.com
```

```
Cannot start service orderer.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/config': mkdir /host_mnt/c/Users/opc/.vscode/
extensions/oracle.oracle-blockchain-1.4.0: operation not permitted
ERROR: for ca.example.com
Cannot start service ca.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/crypto-config/peerOrganizations/org1.example.com/
ca': mkdir /host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0: operation not permitted
Encountered errors while bringing up the project.
ERROR: Error in Chaincode deployment
```

You need to rebuild your local environment:

```
ochain run -b
```

Deploy Your Chaincode to a Remote Oracle Blockchain Platform Network

Once you've instantiated and tested your chaincode project on a local network to ensure it's working as designed, you can deploy it to Oracle Blockchain Platform.

Creating Your Connection Profile

You can download your connection profile from the Oracle Blockchain Platform Cloud or Enterprise instance by navigating to the following location in your instance's console: on the **Developer Tools** tab, **Application Development**, and then **Download the development package**. Unzip the downloaded package. The contents should look similar to:

```
artifacts
    crypto
        ordererOrganizations
        peerOrganizations
network.yaml
```

Retrieve your admin credentials from your instance's console: on the **Network** tab, find your organization in the list. Click the **More Actions** icon for your organization, then select **Export Admin Credential from your organization**.

1. Copy the admin certificate from the admin credentials to `artifacts/crypto/ordererOrganizations/<instance-name>/signcert/<instance-name>-signcert.pem` and `artifacts/crypto/peerOrganizations/<instance-name>/signcert/<instance-name>-signcert.pem`.
2. Copy the admin key from the admin credentials to `artifacts/crypto/ordererOrganizations/<instance-name>/keystore/<instance-name>-key.pem` and `artifacts/crypto/peerOrganizations/<instance-name>/keystore/<instance-name>-key.pem`.

Note:

Keep in mind that if you're placing your connection profile in your project folder, if you're synching this with a content repository such as Github you might accidentally share your private keys. Ensure that everything private is added to your `.gitignore` file.

Log In Using the `ochain login` Command

You need to log in to the Oracle Blockchain Platform instance before deploying. The `ochain login` command will help you to authenticate and log in. It will prompt you asking for your username and password. The username and password are the same as the blockchain instance details.

Note:

The `ochain login` must be run before deploying the chaincode to a remote Oracle Blockchain Platform Cloud instance.

Usage: `ochain login [options]`

The following arguments can be used with this command:

```
my-mac:TSProject myname$ ochain login -h
Usage: login [options]
Login to Oracle Blockchain Cloud Service instance

Options :
    -h, --help           output command usage information
    -0, --debug          enable debug logging
    -P, --project        (optional) Path to Chaincode project to
deploy. If not specified, it defaults to current directory.
    -d, --obp-dev-package (optional) Path to the downloaded and
unzipped Oracle Blockchain Development Package. If not specified, it
defaults to CURRENT_DIRECTORY/obp
    -u, --username       (optional) A user name that has install
chaincode privileges. Contact your administrator for more details.
    -p, --password       (optional) user password.

Examples:
$> ochain login
$> ochain login -u john.doe
S> ochain login -u john.doe -p MyPassword!
```

The `login` command uses the path to your downloaded Oracle Blockchain Development connection profile in the `-d` option. If not specified, it defaults to `<current_project_directory>/obp`

If your login is successful, a message will be returned stating "`Login Successful`".

Deploy

Usage: `ochain deploy [options] [...args]`

Following are the arguments and options taken by the `ochain deploy` command:

```
my-mac:TSProject myname$ ochain deploy -h
Usage: deploy [options] [...args]
Deploy chaincode project to Oracle Blockchain Cloud Service

Arguments:
     [...args]    (optional) Chaincode instantiate arguments. Arguments
should be comma separated.

Options :
    -h, --help               output command usage information
    -0, --debug              enable debug logging
    -P, --project            (optional) Path to Chaincode project to
deploy. If not specified, it defaults to current directory.
    -d, --obp-dev-package    (optional) Path to the downloaded and
unzipped Oracle Blockchain Development Package. If not specified, it
defaults to CURRENT_DIRECTORY/obp
    -c, --channel            (optional) Blockchain Platform channel
to deploy chaincode to. If not specified, defaults to the 'default'
channel.
    -u, --wait               (optional) GRPC wait for ready timeout in
milliseconds.
    -p, --username           (optional) A user name that has install
chaincode privileges. Contact your administrator for more details.
    -v, --userversion        (optional) A user-specified chaincode
version.
                             If a version isn't specified, for a new
chaincode it will start at v1 and then increment to v2, v3, and so on.
                             For an existing chaincode, v1.a will
increment to v1.a1, v1 will increment to v2, and v1.0 will increment
to v1.1.

Examples:
$> ochain deploy -u john.doe -d <path to connection profile> -c
channelname
```

Once the chaincode has successfully deployed to the remote Oracle Blockchain Platform, the log will show that:

- It has successfully installed the chaincode project.
- It has successfully instantiated the chaincode on each peer and the channel.

Updating the Chaincode Project

The upgrade of the chaincode is handled automatically by Blockchain App Builder. After you have made changes to your chaincode, just call the `ochain deploy` command again - this will automatically perform the update for you.

If your update is successful, the log will show

- It has successfully upgraded the chaincode version (for example from version 1.0 to 2.0).
- It has successfully installed the chaincode project.
- It has successfully instantiated the chaincode on each peer and the channel.

Troubleshoot the Deployment

You may encounter the following issues when deploying your chaincode project on Oracle Blockchain Platform:

GRPC timeout error
While deploying to remote Oracle Blockchain Platform network, you may get the following GRPC timeout error:

```
[2020-09-23T18:40:17.923] [ERROR] default - Error Invoking chaincode
"TSProject:" Failed to connect
before the deadline
```

This could be a network issue or GRPC wait timeout expired.

Deploy the chaincode again with `-w <timeout>` option. By default the timeout is set to 10000 ms.

```
> ochain deploy -u idcqa -d /Blockchain/DevTools/bp1/oraclebp1-
instance-info -w 30000
```

Package Your Chaincode Project for Manual Deployment to Oracle Blockchain Platform

You can package your chaincode projects for manual deployment to Oracle Blockchain Platform Cloud or Enterprise Edition.

Usage: `ochain package`

The **Package** function creates a zip file containing only the build and distribution files - the `binary`, `libs`, `node_modules`, and `test` folders from your chaincode project are not included. This zip can be manually uploaded to Oracle Blockchain Platform for deployment.

```
my-mac:~ myname$ ochain package -h
Usage: package [options]
Package and archive an Ochain chaincode project
Options :
    -h, --help             output command usage information
    -D, --debug            enable debug logging
    -p, --project <path>   Path to the Ochain chaincode project to be
packaged. If not specified, it defaults to current directory.
    -o, --out <path>       Path to the generated chaincode archive
file. If not specified, it defaults to current directory.
About:
This CLI command packages and archives an existing chaincode project
Examples:
```

```
$> ochain package --project <Path to the Ochain chaicode project> –out
<Path to the generated chaincode archive file>
```

When the command completes successfully, the location of the package will be returned.

This command takes two optional arguments:

- `--project`
  This option defines the location of the Blockchain App Builder chaincode project to be packaged. If not specified, it defaults to the current directory.
- `--out`
  This option can be used to give the output path of the generated archive file. If not specified, it defaults to the current directory.

Example:

```
> ochain package -p /Blockchain/DevTools/bp1/CC -o /Blockchain/
DevTools/bp1/output

"Your chaincode project has been packaged at /Blockchain/DevTools/bp1/
output/CC.zip"
```

Test Your Chaincode Using the CLI

If your chaincode is running on a network, you can test any of the generated methods. Additionally, If you chose to create the `executeQuery` method during your chaincode development, you can run SQL rich queries if your chaincode is deployed to an Oracle Blockchain Platform network.

- Test Your Chaincode on a Local Hyperledger Fabric Network
- Test Your Chaincode on a Remote Oracle Blockchain Platform Network
- Execute Berkeley DB SQL Rich Queries

Test Your Chaincode on a Local Hyperledger Fabric Network

Once your chaincode project is running on a local network, you can test it.

Open a new shell and navigate to the project directory to interact with your chaincodes. After a chaincode is installed and instantiated, you can submit transactions to the functions inside your chaincode by using the `ochain invoke` and `ochain query` commands.

`ochain invoke`

Usage: `ochain invoke <methodName> <methodArguments>`

The following are arguments and options taken by the `ochain invoke` command:

```
my-mac:TSProject myname$ ochain invoke -h
Usage: invoke [options] <methodName> [...args]
Invoke a Chaincode transaction.

Arguments :
    <methodName>  (required) Name of chaincode method to invoke.
    [...args]     (optional) Chaincode method input parameters if any.
Parameters should be array of JSON strings or a string for single
parameter.

Options:
    -h, --help              output command usage information
    -D, --debug             enable debug logging
    -P, --project           (optional) Path to Chaincode project to
deploy. If not specified, it defaults to current directory.
    -d, --obp-dev-package    (optional) Path to the downloaded and
unzipped Oracle Blockchain Development Package. If not specified, it
defaults to 'CURRENT_DIRECTORY/obp'.
    -c, --channel           (optional) Blockchain Channel to deploy
chaincode too. If not specified, it defaults to the 'default' channel.
    -u, --username          (optional, if -d option is applied) A user
name that has install chaincode privileges. Contact your administrator
for more details.

Examples:
$> ochain invoke <method>
without chaincode initial arguments
$> ochain invoke <method>
{"manufacturerld":"m01","rawMaterialAvailable":9,"productsAvailable":4,"
completionDate":-05-26-2020"}'
for a single parameter
$> ochain invoke <method> ['s01','sl0']
for multiple parameters
```

Mac OSX and Linux

If the method takes one argument, enter it as a string. For example:

```
> ochain invoke createSupplier
'{"supplierId":"s01","rawMaterialAvailable":5,"license":"valid
supplier","expiryDate":"2020-05-30","active":true}'
```

Another example:

```
> ochain invoke getSupplierDetails 's01'
'{"supplierId":"s01","rawMaterialAvailable":5,"license":"valid
supplier","expiryDate":"2020-05-30","active":true}'
```

If the method takes more than one argument, they should be separated by a space. For example:

```
> ochain invoke getSupplierByRange 's01' 's03'
```

If you have embedded assets in your chaincode such as an employee asset which uses an embedded address asset:

```
name: employee
  properties:
     name: employeeId
     type: string
     mandatory: true
     id: true

     name: firstName
     type: string
     validate: max(30)
     mandatory: true

     name: lastName
     type: string
     validate: max(30)
     mandatory: true

     name: age
     type: number
     validate: positive(),min(18)

     name: address
     type: address


name: address

type: embedded

properties:
   name: street
   type: string

   name: city
   type: string

   name: state
   type: string

   name: country
   type: string
```

You would use something similar to the following to invoke the chaincode:

```
> ochain invoke createEmployee '{"employeeID":"e01",
"firstName":"John", "lastName":"Doe",
"age":35, "address":{"street":"Elm Ave", "city":"LA",
"state":"California", "country":"US"}}'
```

Windows

Windows command prompt doesn't accept single quotes ('), so all arguments have to be kept in double quotes ("). Any argument that contains a double quote must be escaped.

For example:

```
> ochain invoke createSupplier
"{\"supplierId\":\"s01\",\"rawMaterialAvailable\":5,\"license\":\"valid
supplier\",\"expiryDate\":\"2020-05-30\",\"active\":true}"
```

If the method takes more than one argument, they should be separated by a space. For example:

```
> ochain invoke getSupplierByRange "s01" "s03"
```

If you have embedded assets in your chaincode such as an employee asset which uses an embedded address asset as shown above, you can use something similar to the following to invoke the chaincode:

```
> ochain invoke createEmployee "{\"employeeID\":\"e01\",
\"firstName\":\"John\",
\"lastName\":\"Doe\", \"age\":35, \"address\":{\"street\":\"Elm Ave\",
\"city\":\"LA\",
\"state\":\"California\", \"country\":\"US\"}}"
```

Validations

The method arguments are validated against the validations specified in the specification file. If any validation fails, errors will be listed in the output.

When it invokes successfully it should display a log similar to:

```
========== Started Invoke Chaincode ==========
[2020-06-23T18:37:54.563] [INFO] default - Successfully sent Proposal
and received ProposalResponse
[2020-06-23T18:37:56.619] [INFO default - The chaincode invoke
transaction has been committed on peer localhost:7051
[2020-06-23T18:37:56.619] [INFO] default - The chaincode invoke
transaction was valid.
[2020-06-23T18:37:56.620] [INFO default - Successfully sent transaction
to the orderer.
[2020-06-23T18:37:56.620] [INFO] default - Successfully invoked method
"createSupplier" on chaincode "TSProject" on channel "mychannel"
[2020-06-23T18:37:56.620] [INFO] default -
========== Finished Invoke Chaincode ==========
```

`ochain query`

Usage: `ochain query <methodName> <methodArguments>`

Following are the arguments and options taken by the `ochain query` command:

```
my-mac:TSProject myname$ ochain query -h
Usage: query [options] <methodName> [...args]
Invoke a Chaincode Query.

Arguments :
    <methodName>     (required) Name of chaincode method to invoke.
    [...args]        (optional) Chaincode method input parameters if
any. Parameters should be array of JSON strings or a string for single
parameter.

Options:
    -h, --help              output command usage information
    -D, --debug             enable debug logging
    -P, --project           (optional) Path to Chaincode project to
deploy. If not specified, it defaults to current directory.
    -d, --obp-dev-package   (optional) Path to the downloaded and
unzipped Oracle Blockchain Development Package. If not specified, it
defaults to 'CURRENT_DIRECTORY/obp'.
    -c, --channel           (optional) Blockchain Channel to deploy
chaincode too. If not specified, it defaults to the 'default' channel.
    -u, --username          (optional, if -d option is applied) A user
name that has install chaincode privileges. Contact your administrator
for more details.

Examples:
$> ochain query <method>
without chaincode initial arguments
$> ochain query <method> s01
for a single parameter
$> ochain query <method> 's01','{"manufacturerId":"m01"}'
for multiple parameters
```

The `ochain query` command follows the same rules of passing `<methodName>` and `<methodArguments>` as `ochain invoke`.

- On Mac OSX and Linux, single quotes can be used and there's no need to escape quotes within arguments.
- On Windows, all arguments must surrounded by double quotes and any quote within an argument must be escaped.

Automatic Install and Instantiate After Update

Whenever you update your chaincode, the changes will be compiled, installed and instantiated automatically when it's deployed to a local network. There is no need to strip down or bring up the local network again. All projects will be automatically compiled and deployed on every change.

Test Your Chaincode on a Remote Oracle Blockchain Platform Network

Once your chaincode project has successfully deployed to your remote Oracle Blockchain Platform network, you can test it as described in Test Your Chaincode on a Local Hyperledger Fabric Network.

You can use the same `ochain invoke` and `ochain query` commands to perform all method transactions on a remote Oracle Blockchain Platform Cloud or Enterprise Edition network; everything supported on the local network is also supported on the remote network. Simply pass the Oracle Blockchain Platform connection profile as an option (`-d`) to the commands.

Example

```
> ochain invoke createSupplier
'{"supplierId":"s01","rawMaterialAvailable":5,"license":"valid
supplier","expiryDate":"2020-05-30","active":true}' -d
/Blockchain/DevTools/bp1/oraclebp1-instance-info
```

Execute Berkeley DB SQL Rich Queries

If you chose to create the `executeQuery` method during your chaincode development, you can run SQL rich queries if your chaincode is deployed to an Oracle Blockchain Platform network.

If you have used `executeQuery` in the `customMethods` section of the specification file, a corresponding `executeQuery` method will be created in the controller.

Specification file:

```
customMethods:
    - executeQuery
    - "fetchRawMaterial(supplierid: string, rawMaterialSupply: number)"
    - "getRawMaterialFromSupplier(manufacturerId: string, supplierId:
string, rawMaterialSupply: number)"
    - "createProducts(manufacturerId: string, rawMaterialConsumed:
number, productsCreated: number)"
    - "sendProductsToDistribution()"
```

Controller file:

```
**
*
* BDB sql rich queries can be executed in OBP CS/EE.
* This method can be invoked only when connected to remote OBP CS/EE
network.
*
*/
@Validator(yup.string())
public async executeQuery(query: string) {
    const result = await OchainController.query(query);
    return result;
}
```

You can invoke this method to execute Berkeley DB SQL rich queries on Oracle Blockchain Platform network, ensuring that you use the `-d` option to specify the location of your downloaded connection profile.

Example:

```
> ochain query executeQuery "SELECT key, valueJson FROM <STATE> WHERE
json_extract(valueJson, '$.rawMaterialAvailable') = 4" -d
/Blockchain/DevTools/bp1/oraclebp1-instance-info
```

The entire SQL query is taken in the argument, so you can make changes to your query on the fly.

Synchronize Specification File Changes With Generated Source Code

You can use this command to bring new changes from the specification file to the current source files (model and controller). This command works with both TypeScript and Go projects.

Note:

- Code sync is unidirectional - you can bring changes from your specification file into your chaincode project, but not the other way around. Changes made in your chaincode project remain as-is after the synching process.
- This command only works if the chaincode project has been scaffolded using a specification file. Ensure you do not delete, rename or move the specification file if you plan to sync any changes from the specification file to the source code in future.
- If you have used a single specification file to generate more than one chaincode project, you can only synchronize one project at a time using this command.

Usage:

```
sync [options] [...args]

my-mac:TsProject myname@ ochain sync -h
Usage: sync [options] [...args]
Synchronize Changes from spec file to the required chaincode.
Arguments:
    [...args] (optional) Sync Arguments.
Options :
    -h, --help              output command usage information
    -D, --debug             enable debug logging
    -p, --project <path>    (optional) Path to Chaincode project to
sync. If not specified, it defaults to current directory
    -c, --confirm <bool>    (optional) Parameter to ensure if you have
resolved all the conflicts, and commit changes
Examples :
$> ochain sync
without chaincode initial arguments
```

This command has two optional arguments:

- `-p / --project`
  This option takes the chaincode project directory where the sync needs to be performed. If not specified, it defaults to the current directory.
- `-c / --confirm`
  This option takes boolean (true/false) values. If there are any conflicts during the merging process, you must resolve those conflicts manually and set this option to true in the next sync cycle. Don't use this option if you're not sure that the conflicts have been merged.

Writing Unit Test Cases and Coverage Reports for the Chaincode Project

Blockchain App Builder includes support for writing unit test cases and coverage reports for the generated chaincode projects.

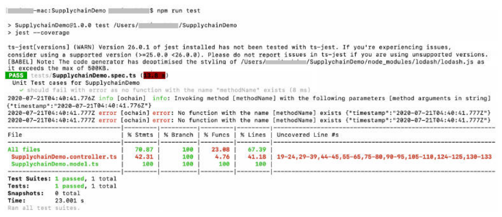
> **Note:**
>
> If you're running your unit tests in VS Code, it can be done in the Terminal window.

TypeScript

To write unit test cases for a TypeScript chaincode, refer to the file `<chaincodeName>.spec.ts` in the `tests` folder inside the generated chaincode project. This file provides the complete unit testing setup for TypeScript chaincodes, and also an example unit test case in the comments section for reference. Following the example, you will be able write unit test cases for your chaincode methods.

This example uses `ChaincodeMockStub` by `@theledger/fabric-mock-stub` for mocking the shim stub. With this mockStub, you can call `mockInit`, `mockInvoke` or direct chaincode methods. Refer to https://github.com/wearetheledger/fabric-mock-stub for more details.

The unit test cases can be run by executing the command `npm run test` from the chaincode project folder. This will also give you the coverage reports.



```
[redacted]-mac:SupplychainDemo [redacted]$ npm run test

> SupplychainDemo@1.0.0 test /Users/[redacted]/SupplychainDemo
> jest --coverage

ts-jest[versions] (WARN) Version 26.0.1 of jest installed has not been tested with ts-jest. If you're experiencing issues,
consider using a supported version (>=25.0.0 <26.0.0). Please do not report issues in ts-jest if you are using unsupported versions.
[BABEL] Note: The code generator has deoptimised the styling of /Users/[redacted]/SupplychainDemo/node_modules/lodash/lodash.js as
it exceeds the max of 500KB.
PASS  tests/SupplychainDemo.spec.ts (13.8 s)
  Unit Test cases for SupplychainDemo
    ✓ should fail with error as no function with the name "methodName" exists (8 ms)
2020-07-21T04:40:41.776Z info [ochain]  info: Invoking method [methodName] with the following parameters [method arguments in string]
{"timestamp":"2020-07-21T04:40:41.776Z"}
2020-07-21T04:40:41.777Z error [ochain] error: No function with the name [methodName] exists {"timestamp":"2020-07-21T04:40:41.777Z"}
2020-07-21T04:40:41.777Z error [ochain] error: No function with the name [methodName] exists {"timestamp":"2020-07-21T04:40:41.777Z"}
-------------------------------|---------|----------|---------|---------|---------------------------------------------------------
File                           | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------------|---------|----------|---------|---------|---------------------------------------------------------
All files                      |   70.87 |      100 |   23.08 |   67.39 |
 SupplychainDemo.controller.ts |   42.31 |      100 |    4.76 |   41.18 | 19-24,29-39,44-45,55-65,75-80,90-95,105-110,124-125,130-133
 SupplychainDemo.model.ts      |     100 |      100 |     100 |     100 |
-------------------------------|---------|----------|---------|---------|---------------------------------------------------------
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        23.001 s
Ran all test suites.
```

Go

To write unit test cases for a Go chaincode, refer to the file `src/src_test.go` inside the generated chaincode project. This file provides the complete unit testing setup for Go chaincodes, and also an example unit test case in the comments section for reference. Following the example, you will be able write unit test cases for your chaincode methods.

The unit test cases can be run by executing the command `go test` from the chaincode project folder. For coverage, add the flag `--cover`.

Example: `go test --cover`.

Using the Blockchain App Builder Extension for Visual Studio Code

The Blockchain App Builder extension for Visual Studio Code helps you build and scaffold a fully-functional chaincode project from a specification file. Once the project is built, you can run and test it on a local Hyperledger Fabric network, or your provisioned Oracle Blockchain Platform network. You can then run SQL rich queries, debug the chaincode, or write and run unit tests using the generated tools.

Table C-3 Workflow When Using the VS Code Extension

Task	Description	More Information
Install and configure	Download the Blockchain App Builder VS Code extension from your Oracle Blockchain Platform console and install it and any prerequisite software.	• Install and Configure the Blockchain App Builder Extension for Visual Studio Code
Create the chaincode project	Create a specification file, and then generate your chaincode project.	• Create a Chaincode Project with the Blockchain App Builder VS Code Extension Detailed reference information about the structure and contents of the specification file and the generated chaincode project: • Input Specification File • Scaffolded TypeScript Chaincode Project • Scaffolded Go Chaincode Project
Deploy the chaincode	After your chaincode project is created, you can deploy it locally to the included pre-configured Hyperledger Fabric network, or remotely to your Oracle Blockchain Platform Cloud or Enterprise Edition. You can also package the chaincode project for manual deployment to Oracle Blockchain Platform.	• Deploy the Chaincode to a Local Hyperledger Fabric Network • Deploy Your Chaincode to a Remote Oracle Blockchain Platform Network • Package Your Chaincode Project for Manual Deployment to Oracle Blockchain Platform
Test the chaincode	Once your chaincode is running on a network, you can test any of the generated methods. Additionally, If you chose to create the `executeQuery` method during your chaincode development, you can run SQL rich queries if your chaincode is deployed to an Oracle Blockchain Platform network.	• Test Your Chaincode on a Local Hyperledger Fabric Network • Testing Lifecycle Operations on a Remote Oracle Blockchain Platform Network • Execute Berkeley DB SQL Rich Queries
Debug the chaincode	You can do line-by-line debugging in Visual Studio Code.	• Debugging from Visual Studio Code

Table C-3 (Cont.) Workflow When Using the VS Code Extension

Task	Description	More Information
Synchronize your updates	When you update your specification file, you can synchronize the changes with the generated chaincode files.	• Synchronize Specification File Changes With Generated Source Code
Running unit tests	A basic unit test case setup is included in the project. Additional tests can be added and run.	• Writing Unit Test Cases and Coverage Reports for the Chaincode Project

Install and Configure the Blockchain App Builder Extension for Visual Studio Code

The Blockchain App Builder extension for Visual Studio Code can be downloaded through the Oracle Blockchain Platform console.

The following platforms are supported:

- Mac OSX
- Oracle Enterprise Linux 7.7 or 7.8
- Windows 10

Prerequisites for OSX and Linux

Before you install Blockchain App Builder CLI on your local system, you must install the prerequisites.

- VS Code version 1.48.0 or later
  The VS Code version can be found by running: `code --version`
- Node v10.18.1 or later, and npm v6.x or later
  The Node version can be found by running: `node --version`

  The npm version can be found by running: `npm --version`

  If installing Node and npm using a manager such as 'nvm' or 'nodenv', you will need to set the default/global version and restart VS Code for the version to be detected by the Prerequisites page.
- Docker version v18.09.0 or greater
  Docker version can be found by running: `docker --version`
- Docker Compose v1.23.0 or greater
  Docker Compose version can be found by running: `docker-compose --version`

The following dependencies are optional:

- Go version v1.14 or later for developing Go chaincodes.
  The Go version can be found by running: `go version`

Install Node and npm using nvm

We suggest using nvm to install Node and npm because it will give you the ability to run more commands without sudo.

1. Install nvm:

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/
   install.sh | bash
   ```

2. Add the below code snippet to `~/.bash_profile`, `~/.profile`, `~/.bashrc` or `~/.zshrc`.

   ```
   export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "$
   {HOME}/.nvm" || printf
   %s "${XDG_CONFIG_HOME}/nvm")"
   [ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
   ```

3. Log out and log back in to your operating system.
4. Verify the nvm installation:

   ```
   nvm version
   ```

5. Install Node and npm:

   ```
   nvm install 10.18.1
   ```

6. Set Node 10.18.1 as the default in nvm:

   ```
   > nvm alias default 10.18.1
   default -> 10.18.1 (-> v10.18.1)
   ```

Prerequisites for Windows

Before you install Blockchain App Builder CLI on your local system, you must install the prerequisites.

- Docker Desktop for Windows v2.x (tested with 2.5.0.1). When prompted by Docker, provide the Filesharing permissions (required for App Builder).
- Node v10.18.1 or later (tested with 10.22.1)
- npm v6.x (tested with 6.13.4)
- Perl v5.x (tested with ActiveState Perl 5.28)
- Install Windows Build Tools in a powershell with administrative access. `npm install --global windows-build-tools`
- If you are developing Go smart contracts, install Go v1.14
- If you want to use the Synchronization feature of App Builder, Git should be installed and your username and email should be configured as follows.

  ```
  > git config --global user.name "<your_name>"
  > git config --global user.email "<email>"
  ```

- Download and build OpenSSL

Download and Build OpenSSL

1. Download OpenSSL from: https://www.openssl.org/source/old/1.0.2/openssl-1.0.2u.tar.gz
2. Unzip the tarball.
3. Open the Visual C++ 2017/2019 Native Tools command prompt. In the Windows search bar, search for `x64 Native Tools Command Prompt for VS`.
4. Navigate to the extracted OpenSSL folder. Run the following commands as an administrator:

```
> perl Configure VC-WIN64A –prefix=C:\OpenSSL-Win64
> ms\do_win64a
> nmake -f ms\ntdll.mak          This can take up to 15 minutes to
complete.
> cd out32dll
> ..\ms\test
> cd ..
> md C:\OpenSSL-Win64
> md C:\OpenSSL-Win64\bin
> md C:\OpenSSL-Win64\lib
> md C:\OpenSSL-Win64\include
> md C:\OpenSSL-Win64\include\openssl
> copy /b inc32\openssl\* C:\OpenSSL-Win64\include\openssl
> copy /b out32dll\ssleay32.lib C:\OpenSSL-Win64\lib
> copy /b out32dll\libeay32.lib C:\OpenSSL-Win64\lib
> copy /b out32dll\ssleay32.dll C:\OpenSSL-Win64\bin
> copy /b out32dll\libeay32.dll C:\OpenSSL-Win64\bin
> copy /b out32dll\openssl.exe C:\OpenSSL-Win64\bin
> copy /b C:\OpenSSL-Win64\bin\libeay32.dll
C:\Windows\System32\libeay32.dll
> copy /b C:\OpenSSL-Win64\bin\ssleay32.dll
C:\Windows\System32\ssleay32.dll
```

Install the Blockchain App Builder Extension

1. Download the extension from your Blockchain Platform console on the **Developer Tools** tab. On the Blockchain App Builder pane, under **Download**, select **Visual Studio Code Extension**.
2. In Visual Studio Code, open the **Extensions** panel and from the **More Actions** menu, select **Install from VSIX**.
3. Locate the downloaded `oracle-ochain-extension-1.4.0.vsix` file and click **Install**.
4. Restart Visual Studio Code to complete installation of the extension.

After the installation, you can use the Oracle Blockchain App Builder icon on the left side of VS Code to open the Blockchain App Builder panel.

Additionally, the Blockchain App Builder command line interface is automatically installed as part of the extension for VS Code if you haven't already installed it separately. Any CLI commands can be run in the VS Code console window.

Create a Chaincode Project with the Blockchain App Builder VS Code Extension

To create a Chaincode Project when using the Blockchain App Builder, you need to scaffold a chaincode project from a detailed specification file. This generates a project with all the files you need.

Background

Blockchain App Builder initializes and scaffolds a chaincode project right out of the box for you. Based on simple input, **Create New Chaincode** can generate complex chaincode projects with features such as:

- Multiple assets (models) and their behaviors (controllers)
- Auto-generate CRUD (Create/Read/Update/Delete) and non-CRUD methods
- Automatic validation of arguments
- Marshalling/unmarshalling of arguments
- Transparent persistence capability (ORM)
- Calling rich queries
- Transient and private data support
- Identity management

The generated project follows model/controller and decorator pattern, which allows an asset's properties maintained on the ledger to be specified as typed fields and extended with specific behaviors and validation rules. This reduces the number of lines of code which helps in readability and scalability.

Create a Specification File

Before you begin, you need to create an input specification file. Note that you cannot alter the sample specification files that were installed as part of Blockchain App Builder, but you can duplicate them or use them as a reference file for your own specification files.

1. In the **Specifications** pane, select **Create New Spec File**.
2. The **Specifications Details** pane opens:
   - Enter the name for the specification file.
   - Select the file type - YAML and JSON are supported.
   - Optionally enter a description for the file.
   - The **Reference File** drop-down allows you to generate your specification file from a pre-existing file in your workspace if you have a file you'd like to use as a template. If nothing is selected, the created file will be empty and you can enter your specification from scratch.
   - Enter the location where you want the specification file to be stored on your system.

   Click **Save**.

The new specification file is created and appears in the **Specifications** pane. Click on it to open it in the editor.

Import a Specification File

If you have a pre-existing specification file, you can import it:

1. In the **Specifications** pane, click **More Actions** and select **Import Specification**.
2. Browse to your file and click **Import Specification**.

The specification file is imported and appears in the **Specifications** pane. Click on it to open it in the editor.

Duplicate a Specification File

You can also duplicate a specification file that's already in your **Specifications** pane by right-clicking it and selecting **Duplicate**. Right-click the file and select **Rename** to update the name.

The details about the contents of specifications files are described here: Input Specification File. Use this information and the sample specification templates to create your specification content.

Scaffolding the Chaincode Project

Once you have a specification file that meets your needs, generate your chaincode project.

1. In the **Chaincodes** pane, select **Create New Chaincode**.
2. The **Chaincode Details** pane opens:
   - Enter the name of your chaincode project
   - Select the language: TypeScript or Go
   - Select the specification file you're using to create the chaincode.
   - Enter the location or Go domain where you want the project to be created within your local development environment.

   Click **Create**.

When your project is created, it will be shown in the **Chaincodes** pane. All the files required by the chaincode will be in the project. For a detailed overview of the files created, see:

- Scaffolded TypeScript Chaincode Project
- Scaffolded Go Chaincode Project

Note:

- The **Chaincodes** pane allows you open and work with content within the chaincode project, but won't let you add, delete, or rename files within the project. To do that, right-click your project and select **Open in Explorer**. This opens the project in the VS Code Explorer view.
- Deleting or renaming files in the chaincode project can potentially break the link between the project files and the specification file used to create it. If you plan to synchronize your code between the two, don't change the file names.

Import an Existing Chaincode Project

If you've created a chaincode project through the CLI or you've cleaned your VS Code blockchain content and want to import a locally saved project, in the **Chaincodes** pane click the **More Actions...** icon and select **Import Chaincode**. Browse to the project and click **Import Chaincode**.

Input Specification File

The Blockchain App Builder initialization command reads the input specification file and generates the scaffolded project with several tools to assist in the chaincode development process.

With the specification file you can specify multiple asset definitions and behavior, CRUD and non-CRUD method declaration, custom methods, validation of arguments, auto marshalling/unmarshalling, transparent persistence capability, and invoking rich data queries using SQL SELECTs or CouchDB Query Language. These features will be generated for you.

The specification file can be written in either `yaml` or `json`. You can see sample specification files in both formats in the Blockchain App Builder package download:

- `fabcar.yml`
- `marbles.yml`

Note:

As per Go conventions, exported names begin with a capital letter. Therefore all the asset properties and methods must have names starting with capital letters in the specification file.

Structure of the Specification File

The specification file should be structured in the following way:

```
assets:
    name:
    properties:
        name:
        type:
        id:
        derived:
           strategy:
           algorithm:
           format:
        mandatory:
        default:
        validate:
    methods:
        crud:
        others:
    type:
customMethods:
```

Table C-4 Specification File Parameter Descriptions and Examples

Entry	Description	Examples
`assets:`	This property takes the definition and behavior of the asset. You can give multiple asset definitions here.	
`name:`	The name of the asset.	`name: owner # Information about the owner`
`properties:`	Describe all the properties of an asset.	
`name:`	The name of the property.	`name: ownerId # Unique ID for each owner`

`type:`	Basic types are supported:
	• `number`
	• `string`
	• `boolean`
	• `date`
	• `array`
	• `embedded`
	For Go chaincodes, `number` is mapped to `init`. Other types such as
	• `float`
	• `complex`
	• `unsigned/ signed int`
	• `8/16/32/64 bits`
	are not supported at this time.
		`name: year        # Model year`
		`type: number`
		`mandatory: true`
		`validate: min(1910),max(2020)`
		`name: color       # Color - no validation as color names are innumerable`
		`type: string`
		`mandatory: true`

Table C-4 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`id:`	• true
	This specifies the identifier of this asset. This property is mandatory.
		```
		name: owner            # Informmation about the
		owner
		properties:
		     name: ownerId    # Unique ID for each owner
		     type: string
		     mandatory: true
		     id: true
		     name: name        # Name of the owner
		     type: string
		     mandatory: true
		```

Table C-4 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`derived:`	This property specifies that the id property is derived from other keys. Dependent properties should be `string` datatype and not an embedded asset.
	
	This property has two mandatory parameters:
	• `strategy`: takes values of `concat` or `hash`.
	• `format`: takes an array of specification strings and values to be used by the strategy.
	
	Example 1:
	• The property `employeeID` is dependent on the `firstName` and `lastName` properties.
	• This property is a concatenation of the values listed in the `format` array.
	• `IND%1#%2%tIND` is the 0th index in the array and describes the final format.
	• `%n` is a position specifier that takes its values from the other indexes in the array.
	• `%t` indicates the value should be `stub.timestamp` from the channel header.
	• If you need to use the character `%` in the `format`
		Example 1
		
		`name: employee`
		`  properties:`
		`     name: employeeId`
		`     type: string`
		`     mandatory: true`
		`     id: true`
		`     derived:`
		`         strategy: concat`
		`         format:`
		`["IND%1#%2%tIND","firstName","lastName"]`
		
		`     name: firstName`
		`     type: string`
		`     validate: max(30)`
		`     mandatory: true`
		
		`     name: lastName`
		`     type: string`
		`     validate: max(30)`
		`     mandatory: true`
		
		`     name: age`
		`     type: number`
		`     validate: positive(),min(18)`
		
		Example 2
		
		`name: account`
		`  properties:`
		`     name: accountId`
		`     type: string`
		`     mandatory: true`
		`     id: true`
		`     derived:`
		`         strategy: hash`
		`         algorithm: 'sha256'`
		`         format:`
		`["IND%1#%2%t","bankName","ifsccode"]`
		
		`     name: bankName`
		`     type: string`
		`     validate: max(30)`
		`     mandatory: true`
		
		`     name: ifsccode`

Table C-4 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

	string, it should be escaped with another %.
	• The final format in this example would be: `INDfirstName#lastName1606885454916IND`
	Example 2:
	• When using `hash`, you must also use the `algorithm` parameter. The default is `sha256`; `md5` is also supported.
	• `IND%1#%2%t` is the 0th index in the array and describes the final format.
	• `%n` is a position specifier that takes its values from the other indexes in the array.
	• `%t` indicates the value should be `stub.timestamp` from the channel header.
	• If you need to use the character % in the `format` string, it should be escaped with another %.
		`type: string`
		`mandatory: true`

Table C-4 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`mandatory:`	• true
	• false
	The corresponding property is mandatory and cannot be skipped while creating an asset.
		`name: phone       # Phone number - validate as (ddd)-ddd-dddd where dashes could also be periods or spaces`
		`type: string`
		`mandatory: true`
		`validate: /^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$/`
		`name: cars        # The list of car VINs owned by this owner`
		`type: string[]`
		`mandatory: false`

`default:`	This gives you the default value of this property.	
`validate:`	The given property is validated against some of the out-of-box validations provided by Blockchain App Builder. You can chain validations if you ensure that the chain is valid. If the `validate` property is not provided, then the validation is done against only the property `type`.	

`validate:`
type: number
	• positive()
	• negative()
	• min()
	• max()
	These validations can be chained together separated by commas.
		`name: offerApplied`
		`type: number`
		`validate: negative(),min(-4)`
		
		`name: year   # Model year`
		`type: number`
		`mandatory: true`
		`validate: min(1910),max(2020)`

Table C-4 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`validate:`
type: string
	• `min()`
	• `max()`
	• `email()`
	• `url()`
	• `/regex/` - supports PHP regex
	For Go chaincodes, regular expressions which contain certain reserved characters or whitespace characters should be properly escaped.
		`name: website`
		`type: string`
		`mandatory: false`
		`validate: url()`
		
		`name: phone  # Phone number - validate as (ddd)-ddd-dddd where dashes could also be periods or spaces`
		`type: string`
		`mandatory: true`
		`validate: /^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$/`
		
		`name: Color #Color can be red, blue, or green`
		`type: string`
		`mandatory: true`
		`validate: /^\\s*(red|blue|green)\\s*$/`

`validate:`
type: boolean
	• true
	• false
	In the example, the validation of property `active` is by the type itself (`boolean`)
		`name: active`
		`type: boolean`

`validate:`
type: array
	By type itself, in the form of `type: number[]`, this conveys that the array is of type number.
	You can enter limits to the array in the format `number[1:5]` which means minimum length is 1, maximum is 5. If either one is avoided, only min/max is considered.
		`name: items`
		`type: number[:5]`

Table C-4 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`validate:`
type: date
	• `min()`
	• `max()`
	Date should be one of these formats:
	• `YYYY-MM-DD`
	• `YYYY-MM-DDTHH:MM:SSZ`, where T separates the date from the time, and the Z indicates UTC. Timezone offsets can replace the Z as in -05:00 for Central Daylight Savings Time.
		`name: expiryDate`
		`type: date`
		`validate: max('2020-06-26')`
		
		`name: completionDate`
		`type: date`
		`validate: min('2020-06-26T02:30:55Z')`

`methods:`	Use this to state which of the CRUD (Create/Read/Update/Delete) or additional methods are to be generated.
	By default, if nothing is entered, all CRUD and other methods are generated.
		`methods:`
		`    crud: [create, getById, update, delete]`
		`    others: [getHistoryById, getByRange]`

`crud:`	• `create`
	• `getByID` (read)
	• `update`
	• `delete`
	If this array is left empty, no CRUD methods will be created.
	If the `crud` parameter is not used at all, all four methods will be created by default.
		`methods:`
		`    crud: [create, getById, delete]`
		`    others: [] # no other methods will be created`

Table C-4 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`others:`	• `getHistoryById`
	• `getByRange`
	`getHistoryById` returns the history of the asset in a list.
	`getByRange` returns all the assets in a given range.
	If this array is left empty, no other methods will be created.
	If the `others` parameter is not used at all, both methods will be created by default.
		```
		methods:
		    crud: [create, delete]
		    others: [] # no other methods will be created
		```
		```
		methods:
		    crud: [create, getById, update, delete]
		    others: [getHistoryById, getByRange]
		```

Table C-4 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`type:`	This attribute if set to `embedded` defines the asset as an embedded asset. Embedded assets do not have CRUD methods and have to be part of another asset to store in the ledger.
	
	In the example, the property `address` is embedded, and is defined in another asset.
		Asset: `employee`
		
		`name: employee`
		`  properties:`
		`     name: employeeId`
		`     type: string`
		`     mandatory: true`
		`     id: true`
		
		`     name: firstName`
		`     type: string`
		`     validate: max(30)`
		`     mandatory: true`
		
		`     name: lastName`
		`     type: string`
		`     validate: max(30)`
		`     mandatory: true`
		
		`     name: age`
		`     type: number`
		`     validate: positive(),min(18)`
		
		`     name: address`
		`     type: address`
		
		Asset: `address`
		
		`name: address`
		
		`type: embedded`
		
		`properties:`
		`   name: street`
		`   type: string`
		
		`   name: city`
		`   type: string`
		
		`   name: state`
		`   type: string`
		
		`   name: country`
		`   type: string`

Table C-4 (Cont.) Specification File Parameter Descriptions and Examples

Entry	Description	Examples

`customMethods:`	This property creates invokable custom method templates in the main controller file. It takes the method signature and creates the function declaration in the controller file.
	
	You can provide language specific function declarations here.
	
	We provide a custom method named `executeQuery`. If it's added to the specification file, it details how Berkeley DB SQL and CouchDB rich queries can be executed. This method can be invoked only when you are connected to Oracle Blockchain Platform Cloud or Enterprise Edition.
		TypeScript
		
		`customMethods:`
		`    - executeQuery`
		`    - "buyCar(vin: string, buyerId: string, sellerId: string, price: number, date: Date)"`
		`    - "addCar(vin: string, dealerId: string, price: number, date: Date)"`
		
		Go
		
		`customMethods:`
		`    - executeQuery`
		`    - "BuyCar(vin string, buyerId string, sellerId string, price int)"`
		`    - "AddCar(vin string, dealerId string, price int)"`

Scaffolded TypeScript Chaincode Project

Blockchain App Builder takes the input from your specification file and generates a fully-functional scaffolded chaincode project.

If the chaincode project uses the TypeScript language, the scaffolded project contains three main files:

- `main.ts`
- `<chaincodeName>.model.ts`
- `<chaincodeName>.controller.ts`

All the necessary libraries are installed and packaged. The `tsconfig.json` file contains the necessary configuration to compile and build the TypeScript project.

The `<chaincodeName>.model.ts` contains multiple asset definitions and `<chaincodeName>.controller.ts` contains the assets behavior and CRUD methods.

The various decorators in `model.ts` and `controller.ts` provide support for features like automatic validation of arguments, marshalling/unmarshalling of arguments, transparent persistence capability (ORM) and calling rich queries.

Reference:

- Asset
- Decorators
- Models
- Controller
- CRUD Methods
- Model Methods
- Controller Method Details
- Custom Methods
- Init Method

Asset

By default every class which extends `OchainModel` will have an additional read-only property called `assetType`. This property can be used to fetch only assets of this type. Any changes to this property are ignored during the creation and updating of the asset. The property value by default is `<chaincodeName>.<assetName>`.

```
@Id('supplierId')
export class Supplier extends OchainModel<Supplier> {
    public readonly assetType = 'tsdeml36.supplier';
    @Mandatory()
    @Validate(yup.string())
    public supplierId: string;
```

Decorators

Class decorators
`@Id(identifier)`

This decorator identifies the property which uniquely defines the underlying asset. This property is used as a key of the record, which represents this asset in the chaincode's state. This decorator is automatically applied when a new TypeScript project is scaffolded. The 'identifier' argument of the decorator takes the value from specification file.

```
@Id('supplierId')
export class Supplier extends OchainModel{
...
}
```

Property decorators
Multiple property decorators can be used. The decorators are resolved in top to bottom order.

```
@Mandatory()
```

This marks the following property as mandatory so it cannot be skipped while saving to the ledger. If skipped it throws an error.

```
@Mandatory()
public supplierID: string;
```

`@Default(param)`

This property can have a default value. The default value in the argument (`param`) is used when the property is skipped while saving to the ledger.

```
@Default('open for business')
@Validate(yup.string())
public remarks: string;
```

`@Validate(param)`

The following property is validated against the schema presented in the parameter. The argument `param` takes a yup schema and many schema methods can be chained together. Many complex validations can be added. Refer to https://www.npmjs.com/package/yup for more details.

```
@Validate(yup.number().min(3))
public productsShipped: number;
```

`@Embedded(PropertyClass)`

This property decorator marks the underlying property as an embeddable asset. It takes the embeddable class as a parameter. This class should extend the `EmbeddedModel` class. This is validated by the decorator.

In this example, `Employee` has a property called `address` of type `Address`, which is to be embedded with the `Employee` asset. This is denoted by the `@Embedded()` decorator.

```
export class Employee extends OchainModel<Employee> {

   public readonly assetType = 'TsSample.employee';

   @Mandatory()
   @Validate(yup.string())
   public emplyeeID: string;

   @Mandatory()
   @Validate(yup.string().max(30))
   public firstName: string;

   @Mandatory()
   @Validate(yup.string().max(30))
   public lastName: string;

   @Validate(yup.number().positive().min(18))
```

```
    public age: number;

    @Embedded(Address)
    public address: Address;
}


export class Address extends EmbeddedModel<Address> {

    @Validate(yup.string())
    public street: string;

    @Validate(yup.string())
    public city: string;

    @Validate(yup.string())
    public state: string;

    @Validate(yup.string())
    public country: string;
}
```

When a new instance of the `Address` class is created, all the properties of the `Address` class are automatically validated by the `@Validate()` decorator. Note that the `Address` class does not have the `assetType` property or `@Id()` class decorator. This asset and its properties are not saved in the ledger separately but are saved along with the `Employee` asset. Embedded assets are user defined classes that function as value types. The instance of this class can only be stored in the ledger as a part of the containing object (`OchainModel` assets). All the above decorators are applied automatically based on the input file while scaffolding the project.

`@Derived(STRATEGY, ALGORITHM, FORMAT)`

This decorator is used for defining the attribute derived from other properties. This decorator has two mandatory parameters:

- `STRATEGY`: takes values of `CONCAT` or `HASH`. Requires an additional parameter `ALGORITHM` if `HASH` is selected. The default algorithm is `sha256`; `md5` is also supported.
- `FORMAT`: takes an array of specification strings and values to be used by the strategy.

```
@Id('supplierID')
export class Supplier extends OchainModel<Supplier> {

    public readonly assetType = 'chaincodeTS.supplier';

    @Mandatory()
    @Derived(STRATEGY.HASH.'sha256',['IND%1IND%2','license','name'])
    @Validate(yup.string())
    public supplierID: string;
```

```
   @Validate(yup.string().min(2).max(4))
   public license: string;

   @Validate(yup.string().min(2).max(4))
   public name: string;
```

Method decorators
`@Validator(…params)`

This decorator is applied on methods of the main controller class. This decorator is important for parsing the arguments, validating against all the property decorators and returning a model/type object. It takes multiple user created models or yup schemas as parameter.

Note the order of the parameters should be exactly the same as the order of the arguments in the method.

In this example, the `Supplier` model reference is passed in parameters which corresponds to the `asset` type in the method argument. The decorator in run-time would parse and convert the method argument to JSON object, validate against the `Supplier` validators, and upon successful validation convert the JSON object to `Supplier` object and assign it to the `asset` variable. On completion the underlying method is then finally called.

```
@Validator(Supplier)
public async createSupplier(asset: Supplier) {
    return await asset.save();
}
```

In this example, multiple asset references are passed; they corresponds to the object types of the method arguments. Notice the order in the parameters.

```
@Validator(Supplier, Manufacturer)
public async createProducts(supplier: Supplier, manufacturer: 
Manufacturer) {
}
```

Apart from asset reference, yup schema objects could also be passed if the arguments are of basic-types. In this example, `supplierId` and `rawMaterialSupply` are of type `string` and `number` respectively, so the yup schema of similar type and correct order is passed to the decorator. Notice the chaining of yup schema methods.

```
@Validator(yup.string(), yup.number().positive())
public async fetchRawMaterial(supplierID:string, rawMaterialSupply: 
number) {
    const supplier = await Supplier.get(supplierID);
    supplier.rawMaterialAvailable = supplier.rawMaterialAvailable + 
rawMaterialSupply;
    return await supplier.update();
}
```

Models

Every model class extends `OchainModel`. Transparent Persistence Capability or simplified ORM is captured in the `OchainModel` class. If your model needs to need call any of the below ORM methods, you should extend the `OchainModel` class.

ORM methods which are exposed via `OchainModel`:

- `save` – this calls the Hyperledger Fabric `putState` method
- `get` – this calls the Hyperledger Fabric `getState` method
- `update` – this calls the Hyperledger Fabric `putState` method
- `delete` – this calls the Hyperledger Fabric `deleteState` method
- `history` – this calls the Hyperledger Fabric `getHistoryForKey` method
- `getByRange` – this calls the Hyperledger Fabric `getStateByRange` method

See: Model Methods.

Controller

Main controller class extends `OchainController`. There is only one main controller.

```
export class TSProjectController extends OchainController{
```

You can create any number of classes, functions, or files, but only those methods that are defined within the main controller class are invokable from outside, the rest of them are hidden.

CRUD Methods

As described in Input Specification File, you can specify which CRUD methods you want generated in the specification file. For example, if you selected to generate all methods, the result would be similar to:

```
@Validator(Supplier)
public asynch createSupplier(asset: Supplier){
    return await asset.save();
}
public asynch getSupplierById(id: string){
    const asset = await Supplier.get(id);
    return asset;
}
@Validator(Supplier)
public asynch updateSupplier(asset: Supplier){
    return await asset.update();
}
public asynch deleteSupplier(id: string){
    const result = await Supplier.delete(id);
    return result;
}
public asynch getSupplierHistoryById(id: string){
    const result = await Supplier.history(id);
    return result;
}
```

```
@Validator(yup.string(), yup.string())
public asynch getSupplierByRange(startId: string, endId: string){
    const result = await Supplier.getByRange(startId, endId);
    return result;
}
```

Model Methods

save
The `save` method adds the caller `asset` details to the ledger.

This method calls the Hyperledger Fabric `putState` internally. All marshalling/ unmarshalling is handled internally.

```
<Asset>.save(extraMetadata?: any): Promise<any>
```

Parameters:

- `extraMetadata : any` (optional) – To save metadata apart from the asset into the ledger.

Returns:

- `Promise<any>` - Returns a promise on completion

Example:

```
@Validator(Supplier)
public async createSupplier(asset: Supplier) {
    return await asset.save();
}
```

get
The `get` method is a static method of `OchainModel` class which is inherited by the concrete model classes of `{chaincodeName}.model.ts`.

This returns an asset of `<Asset>` if `id` is found in the ledger and has the same type as `<Asset>`. This method calls the Hyperledger Fabric `getState` method internally. Even though any asset with given`id` is returned from the ledger, our method will take care of casting into the caller `Model` type.

If you would like to return any asset by the given `id`, use the generic controller method `getAssetById`.

```
<Asset>.get(id: string): Promise<asset>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise: <Asset>` - Returns object of type `<Asset>`. Even though any asset with given `id` is returned from the ledger, this method will take care of casting into the caller `Asset` type. If the asset returned from the ledger is not of the `Asset` type, then it throws an error. This check is done by the read-only `assetType` property in the `Model` class.

Example:

```
public async getSupplierById: string) {
    const asset = await Supplier.get(id);
    return asset;
}
```

In the example, `asset` is of the type `Supplier`.

`update`
The `update` method updates the caller `asset` details in the ledger. This method returns a promise.

This method calls the Hyperledger Fabric `putState` internally. All the marshalling/unmarshalling is handled internally.

```
<Asset>.update(extraMetadata?: any): Promise<any>
```

Parameters:

- `extraMetadata : any` (optional) – To save metadata apart from the asset into the ledger.

Returns:

- `Promise<any>` - Returns a promise on completion

Example:

```
@Validator(Supplier)
public async updateSupplier(asset: Supplier) {
    return await asset.update();
}
```

`delete`
This deletes the asset from the ledger given by `id` if it exists. This method calls the Hyperledger Fabric `deleteState` method internally.

The `delete` method is a static method of `OchainModel` class which is inherited by the concrete `Model` classes of `{chaincodeName}.model.ts`.

```
<Asset>. delete(id: string): Promise<any>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise <any>` - Returns a promise on completion.

Example:

```
public async deleteSupplier(id: string) {
    const result = await Supplier.delete(id);
    return result;
}
```

`history`
The history method is a static method of `OchainModel` class which is inherited by the concrete `Model` classes of `{chaincodeName}.model.ts`. This returns the asset history given by `id` from the ledger, if it exists.

This method calls the Hyperledger Fabric `getHistoryForKey` method internally.

```
<Asset>.history(id: string): Promise<any[]>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise <any[]>` - Returns any [] on completion.

Example

```
public async getSupplierHistoryById(id: string) {
    const result = await Supplier.history(id);
    return result;
}
```

Example of the returned asset history for `getSupplierHistoryById`:

```
[
    {
        "trxId":
"8ef4eae6389e9d592a475c47d7d9fe6253618ca3ae0bcf77b5de57be6d6c3829",
        "timeStamp": 1602568005,
        "isDelete": false,
        "value": {
            "assetType": "supp.supplier",
            "supplierId": "s01",
            "rawMaterialAvailable": 10,
            "license": "abcdabcdabcd",
            "expiryDate": "2020-05-28T18:30:00.000Z",
            "active": true
        }
    },
    {
        "trxId":
"92c772ce41ab75aec2c05d17d7ca9238ce85c33795308296eabfd41ad34e1499",
        "timeStamp": 1602568147,
        "isDelete": false,
        "value": {
            "assetType": "supp.supplier",
            "supplierId": "s01",
            "rawMaterialAvailable": 15,
            "license": "valid license",
            "expiryDate": "2020-05-28T18:30:00.000Z",
            "active": true
        }
    }
]
```

getByRange
The `getByRange` method is a static method of `OchainModel` class which is inherited by the concrete `Model` classes of `{chaincodeName}.model.ts`.

This returns a list of asset between the range `startId` and `endId`. This method calls the Hyperledger Fabric `getStateByRange` method internally.

Even though any asset with given `id` is returned from the ledger, our method will take care of casting into the caller `Model` type. In above example, `result` array is of the type `Supplier`. If the asset returned from ledger is not of the `Model` type, then it will not be included in the list. This check is done by the read-only `assetType` property in the `Model` class.

If you would like to return all the assets between the range `startId` and `endId`, use the generic controller method `getAssetsByRange`.

```
<Asset>.getByRange(startId: string, endId: string): Promise<Asset[]>
```

Parameters:

- `startId : string` – Starting key of the range. Included in the range.
- `endId : string` – Ending key of the range. Excluded of the range.

Returns:

- `Promise< Asset[ ] >` - Returns array of `<Asset>` on completion.

Example:

```
@Validator(yup.string(), yup.string())
public async getSupplierByRange(startId: string, endId: string){
    const result = await Supplier.getByRange(startId, endId);
    return result;
}
```

`getId`
When the asset has a derived key as `Id`, you can use this method to get a derived ID. This method will return an error if the derived key contains `%t` (timestamp).

Parameters:

- `object` – Object should contain all the properties on which the derived key is dependent.

Returns:

- Returns the derived key as a string.

Example:

```
@Validator(yup.string(), yup.string())
public async customGetterForSupplier(license: string, name: string){
    let object = {
      license : license,
      name: name
    }
    const id = await Supplier.getID(object);
    return Supplier.get(id);
}
```

Controller Method Details

Apart from the above model CRUD and non-CRUD methods, Blockchain App Builder provides out-of-the box support for other Hyperledger Fabric methods from our controller. These methods are:

- `getAssetById`
- `getAssetsByRange`
- `getAssetHistoryById`
- `query`
- `generateCompositeKey`
- `getByCompositeKey`
- `getTransactionId`
- `getTransactionTimestamp`
- `getTransactionInvoker`
- `getChannelID`
- `getCreator`
- `getSignedProposal`
- `getArgs`
- `getStringArgs`
- `getMspID`
- `getNetworkStub`

These methods are available with `this` context itself in the `Controller` class. For example:

```
public async getModelById(id: string) {
    const asset = await this.getAssetById(id);
    return asset;
}
@Validator(yup.string(), yup.string())
public async getModelsByRange(startId: string, endId: string) {
    const asset = await this.getAssetsByRange(startId, endId);
    return asset;
}
public async getModelHistoryById(id: string) {
    const result = await this.getAssetHistoryById(id);
    return result;
}
```

`getAssetById`
The `getAssetById` method returns asset based on `id` provided. This is a generic method and be used to get asset of any type.

```
this< OchainController>.getAssetById(id: string): Promise<byte[]>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise <byte [ ]>` - Returns promise on completion. You have to convert `byte[]` into an object.

`getAssetsByRange`
The `getAssetsByRange` method returns all assets present from `startId` (inclusive) to `endId` (exclusive) irrespective of asset types. This is a generic method and can be used to get assets of any type.

```
this<OchainController>.getAssetsByRange(startId: string, endId:
string):
Promise<shim.Iterators.StateQueryIterator>
```

Parameters:

- `startId : string` – Starting key of the range. Included in the range.
- `endId : string` – Ending key of the range. Excluded of the range.

Returns:

- `Promise< shim.Iterators.StateQueryIterator>` - Returns an iterator on completion. You have to iterate over it.

`getAssetHistoryById`
The `getAssetHistoryById` method returns history iterator of an asset for `id` provided.

```
this<OchainController>.getAssetHistoryById(id: string):
Promise<shim.Iterators.HistoryQueryIterator>
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Promise<shim.Iterators.HistoryQueryIterator>` - Returns a history query iterator. You have to iterate over it.

`query`
The `query` method will run a Rich SQL/Couch DB query over the ledger. This method is only supported for remote deployment on Oracle Blockchain Platform. This is a generic method for executing SQL queries on the ledger.

```
this<OchainController>.query(queryStr: string):
Promise<shim.Iterators.StateQueryIterator>
```

Parameters:

- `queryStr : string` - Rich SQL/Couch DB query.

Returns:

- `Promise<shim.Iterators.StateQueryIterator>` - Returns a state query iterator. You have to iterate over it.

`generateCompositeKey`
This method generates and returns the composite key based on the `indexName` and the attributes given in the arguments.

```
this<OchainController>.generateCompositeKey(indexName: string,
attributes:
string[]): string
```

Parameters:

- `indexName : string` - Object Type of the key used to save data into the ledger.
- `attributes: string[ ]` - Attributes based on which composite key will be formed.

Returns:

- `string` - Returns a composite key.

`getByCompositeKey`
This method returns the asset that matches the key and the column given in the attribute parameter while creating composite key. `indexOfId` parameter indicates the index of the key returned in the array of stub method `SplitCompositeKey`.

Internally this method calls Hyperledger Fabric's `getStateByPartialCompositeKey`, `splitCompositeKey` and `getState`.

```
this<OchainController>.getByCompositeKey(key: string, columns:
string[],
indexOfId: number): Promise<any []>
```

Parameters:

- `key: string` – Key used to save data into ledger.
- `columns: string[ ]` - Attributes based on key is generated.
- `indexOfId: number` - Index of attribute to be retrieved from Key.

Returns:

- `Promise< any [ ]` - Returns any [] on completion.

`getTransactionId`
Returns the transaction ID for the current chaincode invocation request. The transaction ID uniquely identifies the transaction within the scope of the channel.

```
this<OchainController>.getTransactionId(): string
```

Parameters:

- none

Returns:

- `string` - Returns the transaction ID for the current chaincode invocation request.

`getTransactionTimestamp`
Returns the timestamp when the transaction was created. This is taken from the transaction `ChannelHeader`, therefore it will indicate the client's timestamp, and will have the same value across all endorsers.

```
this<OchainController>.getTransactionTimestamp(): Timestamp
```

Parameters:

- `id : string` – Key used to save data into the ledger.

Returns:

- `Timestamp` - Returns the timestamp when the transaction was created.

`getTransactionInvoker`
Returns the caller of the transaction from the Transient map property `bcsRestClientId`.

```
this<OchainController>.getTransactionInvoker(): string
```

Parameters:

- none

Returns:

- `string` - Returns the caller of the transaction.

`getChannelID`
Returns the channel ID for the proposal for chaincode to process.

```
this<OchainController>.getChannelID(): string
```

Parameters:

- none

Returns:

- `string` - Returns the channel ID.

`getCreator`
Returns the identity object of the chaincode invocation's submitter.

```
this<OchainController>.getCreator(): shim.SerializedIdentity
```

Parameters:

- none

Returns:

- `shim.SerializedIdentity` - Returns identity object.

`getSignedProposal`
Returns a fully decoded object of the signed transaction proposal.

```
this<OchainController>.getSignedProposal():
shim.ChaincodeProposal.SignedProposal
```

Parameters:

- none

Returns:

- `shim.ChaincodeProposal.SignedProposal` - Returns decoded object of the signed transaction proposal.

`getArgs`
Returns the arguments as array of strings from the chaincode invocation request.

```
this<OchainController>.getArgs(): string[]
```

Parameters:

- none

Returns:

- `string [ ]` - Returns arguments as array of strings from the chaincode invocation.

`getStringArgs`
Returns the arguments as array of strings from the chaincode invocation request.

```
this<OchainController>.getStringArgs(): string[]
```

Parameters:

- none

Returns:

- `string [ ]` - Returns arguments as array of strings from the chaincode invocation.

`getMspID`
Returns the MSP ID of the invoking identity.

```
this<OchainController>.getMspID(): string
```

Parameters:

- none

Returns:

- `string` - Returns the MSP ID of the invoking identity.

`getNetworkStub`
The user can get access to the shim stub by calling `getNetworkStub` method. This will help user to write its own implementation of working directly with the assets.

```
this<OchainController>.getNetworkStub(): shim.ChaincodeStub
```

Parameters:

- none

Returns:

- `shim.ChaincodeStub` - Returns chaincode network stub.

Custom Methods

The following custom methods were generated from our example specification file.

The `executeQuery` shows how SQL rich queries can be called. The validators against the arguments are added automatically by Blockchain App Builder based on the type of the argument specified in the specification file.

```
/**
*
*    BDB sql rich queries can be executed in OBP CS/EE.
*    This method can be invoked only when connected to remote OBP CS/EE
network.
*
*/
@Validator(yup.string()}
public async executeQuery(query: string) {
    const result = await OchainController.query(query);
    return result;
}
@Validator(yup.string(), yup.number()}
public async fetchRawMaterial(supplierId: string, rawMaterialSupply:
number) {
}
@Validator(yup.string(), yup.string(), yup.number())
public async getRawMaterialFromSupplier(manufacturerId: string,
supplierId: string, rawMaterialSupply: number) {
}
@Validator(yup.string(), yup.number(), yup.number())
public async createProducts(manufacturerId: string,
rawMaterialConsumed: number, productsCreated: number) {
}
public async sendProductsToDistribution() {
}
```

Init Method

We have provided one `init` method in the controller with an empty definition. This method will be called by the Hyperledger Fabric `Init` method during first time instantiating or upgrading the chaincode.

```
export class TestTsProjectController extends OchainController {
    public async init(params: any) {
        return;
}
```

If you would like to initialize any application state at this point, you can use this method to do that.

Scaffolded Go Chaincode Project

Blockchain App Builder takes the input from your specification file and generates a fully-functional scaffolded chaincode project.

If the chaincode project is in the Go language, the scaffolded project contains three main files:

- `main.go`
- `<chaincodeName>.model.go`
- `<chaincodeName>.controller.go`

All the necessary libraries are installed and packaged.

The `<chaincodeName>.model.go` contains multiple asset definitions and `<chaincodeName>.controller.go` contains the asset's behavior and CRUD methods. The various Go struct tags and packages in `model.go` and `controller.go` provide support for features like automatic validation of arguments, marshalling/unmarshalling of arguments, transparent persistence capability (ORM) and calling rich queries.

The scaffolded project can be found in `$GOPATH/src/example.com/<chaincodeName>`

Reference:

- Validators
- Model
- Composite Key Methods
- Stub Method
- Other Methods
- Utility Package
- Controller
- CRUD Methods
- Custom Methods
- Init Method

Validators

Id
`id:"true"`

This validator identifies the property which uniquely defines the underlying model. The asset is saved by the value in this key. This validator automatically applies when a new Go project is scaffolded.

In the below screenshot `"SupplierId"` is the key for the supplier asset and has a tag property `id:"true"` for the SupplierId property.

```
type Supplier struct {
    Supplierld              string        'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable    int           'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License                 string        'json:"License"
validate:"string,min=10"'
    ExpiryDate              date.Date     'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                  bool          'json:"Active"
validate:"bool" default :"true"'
    Metadata                interface{}
'json:"Metadata,omitempty"'
}
```

Derived
`derived:"strategy,algorithm,format"`

This decorator is used for defining the attribute derived from other properties. This decorator has two mandatory parameters:

- `strategy`: takes values of `concat` or `hash`. Requires an additional parameter `algorithm` if `hash` is selected. The default algorithm is `sha256`; `md5` is also supported.
- `format`: takes an array of specification strings and values to be used by the strategy.

```
type Supplier struct{
   AssetType string 'json:"AssetType" final:"chaincode1.Supplier"'
   SupplierId   string  'json:"SupplierId"
validate:"string" id:"true" mandatory:"true"
derived:"strategy=hash,algorith=sha256,format=IND%1%2,License,Name"'
   Name         string  'json:"Name" validate:"string,min=2,max=4"'
   License      string  'json:"License" validate:"string,min=2,max=4"'
}
```

Mandatory
`validate:"mandatory"`

This marks the following property as mandatory and cannot be skipped while saving to the ledger. If skipped it throws an error. In the below example, "`SupplierId`" has a `validate:"mandatory"` tag.

```
Type Supplier struct {
    Supplierld              string      'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable    int         'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License                 string      'json:"License"
validate:"string,min=10"'
    ExpiryDate              date.Date   'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                  bool        'json:"Active" validate:"bool"
default :"true"'
    Metadata                interface{} 'json:"Metadata,omitempty"'
}
```

Default
`default:"<param>"`

This states that the following property can have a default value. The default value in the default tag is used when the property is skipped while saving to the ledger. In the below example property, `Active` has a default value of `true`, provided as tag `default:"true"`

```
Type Supplier struct {
    Supplierld            string       'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License               string       'json:"License"
validate:"string,min=10"'
    ExpiryDate            date.Date    'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                bool         'json:"Active" validate:"bool"
default :"true"'
    Metadata              interface{}  'json:"Metadata,omitempty"'
}
```

Validate types
Basic Go types are validated for a property by defining a validate tag. These are the validate tags based on types:

- string: `validate: "string"`
- date: `validate: "date"`
- number: `validate: "int"`
- boolean: `validate: "bool"`

Min validator
`validate:"min=<param>"`

Using the min validator, minimum value can be set for a property of type number and string.

For type int: In the example, `RawMaterialAvailable` property has a minimum value of 0 and if a value less than 0 is applied to `RawMaterialAvailable` an error will be returned.

For type string: For the string type minimum validator will check the length of the string with the provided value. Therefore, in the below example the `License` property has to be minimum 10 characters long.

Example:

```
Type Supplier struct {
    Supplierld            string       'json:"Supplierld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License               string       'json:"License"
validate:"string,min=10"'
    ExpiryDate            date.Date    'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                bool         'json:"Active" validate:"bool"
default :"true"'
    Metadata              interface{}  'json:"Metadata,omitempty"'
}
```

Max validator
`validate:"max=<param>"`

Using the max validator, the maximum value can be set for a property of type number and string.

For type int: Like the min validator, for type int, if a value provided for the `structfield` is greater than the value provided in the validator then an error will be returned.

For type string: Like the min validator, max validator will also check the length of the string with given value. In the example, the `Domian` property has a maximum value of 50, so if the `Domain` property has a string length more than 50 characters, then an error message will be returned.

```
type Retailer struct {
    Retailerld         string        'json:"Retailerld"
validate:"string,mandatory" id:"true"'
    ProductsOrdered    int           'json:"ProductsOrdered"
validate:"int,mandatory"'
    ProductsAvailable  int           'json:"ProductsAvailable"
validate:"int" default:"1"'
    ProductsSold       int           'json:"ProductsSold"
validate:"int"'
    Remarks            string        'json:"Remarks" validate:"string"
default :"open for business"'
    Items              []int         'json:"Items"
validate:"array=int,range=l-5"'
    Domain             string        'json:"Domain"
validate:"url,min=30,max=50"'
    Metadata           interface{}   'json:"Metadata,omitempty"'
}
```

Date validators
Before validator:

`validate:"before=<param>"`

The before validator validates a property of type `date` to have a value less than the specified in parameter.

In this example, the `ExpiryDate` property should be before `"2020-06-26"` and if not it will return an error.

```
Type Supplier struct {
    Supplierld            string       'json:"Supplierld"
validate:"string,mandatory" id:"true"'
```

```
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,min=0"'
    License               string       'json:"License"
validate:"string,min=10"'
    ExpiryDate            date.Date    'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
    Active                bool         'json:"Active" validate:"bool"
default :"true"'
    Metadata              interface{}  'json:"Metadata,omitempty"'
}
```

After validator:

```
validate:"after=<param>"
```

The before validator validates a property of type `date` to have a value greater than the specified in parameter.

In this example, the `CompletionDate` property should be after `"2020-06-26"` and if not it will return an error.

```
Type Supplier struct {
    Manufacturerld        string       'json:"Manufacturerld"
validate:"string,mandatory" id:"true"'
    RawMaterialAvailable  int          'json:"RawMaterialAvailable"
validate:"int,max=8"'
    ProductsAvailable     int          'json:"ProductsAvailable"
validate:"int"'
    CompletionDate        date.Date    'json:"CompletionDate"
validate:"date,after=2020-06-26"'
    Metadata              interface{}  'json:"Metadata,omitempty"'
}
```

URL validator
`validate:"url"`

The URL validator will validate a property for URL strings.

In this example, the `Domain` property has to be a valid URL.

```
type Retailer struct {
    Retailerld         string        'json:"Retailerld"
validate:"string,mandatory" id:"true"'
    ProductsOrdered    int           'json:"ProductsOrdered"
validate:"int,mandatory"'
    ProductsAvailable  int           'json:"ProductsAvailable"
validate:"int" default:"1"'
    ProductsSold       int           'json:"ProductsSold"
validate:"int"'
    Remarks            string        'json:"Remarks" validate:"string"
default :"open for business"'
    Items              []int         'json:"Items"
validate:"array=int,range=l-5"'
    Domain              string        'json:"Domain"
validate:"string,url,min=30,max=50"'
    Metadata            interface{}   'json:"Metadata,omitempty"'
}
```

Regexp validator
`validate:"regexp=<param>"`

Regexp validator will validate property for the input regular expression.

In this example, the `PhoneNumber` property will validate for a mobile number as per the regular expression.

```
type Customer struct {
Customerld      string       'json:"Customerld"
validate:"string,mandatory" id:"true"'
Name            string       'json:"Name" validate:"string,mandatory"'
ProductsBought  int          'json:"ProductsBought" validate:"int"'
OfferApplied    int          'json:"OfferApplied"
validate :"int,nax=0"'
PhoneNumber     string
'json:"PhoneNumber" validate:"string,regexp=A\(?([0-9]{3})\)?[-. ]?
([0-9]{3})[-. ]?([0-9]{4})$"'
Received        bool         'json:"Received" validate:"bool“'
Metadata        interface{}  'json:"Metadata,omitempty"'
}
```

Multiple validators
Multiple validators can be applied a property.

In this example, the `Domain` property has validation for a string, URL, and min and max string length.

```
type Retailer struct {
    Retailerld          string        'json:"Retailerld"
validate:"string,mandatory" id:"true"'
    ProductsOrdered     int           'json:"ProductsOrdered"
validate:"int,mandatory"'
    ProductsAvailable   int           'json:"ProductsAvailable"
validate:"int" default:"1"'
    ProductsSold        int           'json:"ProductsSold"
validate:"int"'
    Remarks             string        'json:"Remarks" validate:"string"
default :"open for business"'
    Items               []int         'json:"Items"
validate:"array=int,range=l-5"'
    Domain              string        'json:"Domain"
validate:"string,url,min=30,max=50"'
    Metadata            interface{}   'json:"Metadata,omitempty"'
}
```

Model

Asset Type Property

By default every struct will have an additional property called `AssetType`. This property can be useful in fetching only assets of this type. Any changes to this property is ignored during create and update of asset. The property value by default is `<chaincodeName>.<modelName>`.

```
type Supplier struct {
AssetType string 'json:"AssetType" default:"TestGoProject.Supplier"'

SupplierId            string      'json:"SupplierId"
validate:"string,mandatory" id:"true'
RawMaterialAvailable  int         'json:"RawMaterialAvailable"
validate:"int,min=0"'
License               string      'json:"License"
validate:"string,min=10"'
ExpiryDate            date.Date   'json:"ExpiryDate"
validate:"date,before=2020-06-26"'
Active                bool      'json:"Active" validate:"bool"
default:"true"'
Metadata              interface{} 'json:"Metadata,omitempty"'
}
```

ORM

Go chaincodes implement Transparent Persistence Capability (ORM) with the model package.

The following ORM methods are exposed via the model package:

`model.Get`
Queries the ledger for the stored asset based on the given ID.

```
func Get(Id string, result ...interface{}) (interface{}, error)
```

Parameters:

- `Id` - The ID of the asset which is required from the ledger.
- `result (interface{})` - This is an empty asset object of a particular type, which is passed by reference. This object will contain the result from this method. To be used only if type-specific result is required.
- `asset (interface)` - Empty asset object, which is passed by reference. This object will contain the result from this method. To be used only if type-specific result is required.

Returns:

- `interface {}` - Interface contains the asset in the form of `map[string]interface{}`. Before operating on this map, it is required to assert the obtained interface with type `map[string]interface{}`. To convert this map into an asset object, you can use the utility API `util.ConvertMaptoStruct` (see: Utility Package).

- `error` - Contains an error if returned, or is nil.

`model.Update`
Updates the provided asset in the ledger with the new values.

```
func Update(args ...interface{}) (interface{}, error)
```

Parameters:

- `obj (interface)` - The object that is required to be updated in the ledger is passed by reference into this API with the new values. The input asset is validated and verified according to the struct tags mentioned in the model specification and then stored into the ledger.

Returns:

- `interface{}` - The saved asset is returned as an interface.
- `error` - Contains an error if returned, or is nil.

`model.Save`
Saves the asset to the ledger after validating on all the struct tags.

```
func Save(args ...interface{}) (interface{}, error)
```

Parameters:

- `obj/args[0] (interface{})` - The object that needs to be stored in the ledger is passed by reference in this utility method.
- `metadata/args[1] (interface{})` - This parameter is optional. It has been given in order to facilitate you if you're required to store any metadata into the ledger along with the asset at the runtime. This parameter can be skipped if no such requirement exists.

Returns:

- `interface {}` - The asset is returned as an interface.
- `error` - Contains an error if returned, or is nil.

`model.Delete`
Deletes the asset from the ledger.

```
func Delete(Id string) (interface{}, error)
```

Parameters:

- `id (string)` - The ID of the asset which is required to be deleted from the ledger.

Returns:

- `interface {}` - Contains the asset being deleted in the form of `map[string]interface{}`.

`model.GetByRange`
Returns the list of assets by range of IDs.

```
func GetByRange(startKey string, endKey string, asset ...interface{})
([]map[string]interface{}, error)
```

Parameters:

- `startkey (string)` - Starting ID for the range of objects which are required.
- `endkey (string)` - End of the range of objects which are required.
- `asset interface` - (optional) Empty array of assets, which is passed by reference. This array will contain the result from this method. To be used if type-specific result is required.

Returns:

- `[]map[string]interface{}` - This array contains the list of assets obtained from the ledger. You can access the objects iterating over this array and asserting the objects as `map[string]interface{}` and using utility to convert to asset object.
- `error` - Contains an error if returned, or is nil.

`model.GetHistoryById`
Returns the history of the asset with the given ID.

```
func GetHistoryByID(Id string) ([]interface{}, error)
```

Parameters:

- `Id (string)` - ID of the asset for which the history is needed.

Returns:

- `[]interface{}` - This slice contains the history of the asset obtained from the ledger in form of slice of `map[string]interface{}`. You can access each history element by iterating over this slice and asserting the objects as `map[string]interface{}` and using utility to convert to asset object.
- `error` - Contains the error if observed.

`model.Query`
The query method will run a SQL/Couch DB query over the ledger. This method is only supported for remote deployment on Oracle Blockchain Platform. This is a generic method for executing SQL queries on the ledger.

```
func Query(queryString string) ([]interface{}, error)
```

Parameters:

- `queryString (string)` - Input the query string.

Returns:

- `[]interface{}` - This will contain the output of the query. The result is in form of slice of interfaces. You need to iterate over the slice and use the elements by converting them to proper types.
- `error` - Contains the error if observed.

Composite Key Methods

`model.GenerateCompositeKey`
This method generates and returns the composite key based on the `indexName` and the attributes given in the arguments.

```
func GenerateCompositeKey(indexName string, attributes []string)
(string, error)
```

Parameters:

- `indexName (string)` - Object type of the composite key.
- `attrbutes ([]string)` - Attributes of the asset based on which the composite key has to be formed.

Returns:

- `string` - This contains the composite key result.
- `error` - Contains the error if observed.

`model.GetByCompositeKey`
This method returns the asset that matches the key and the column given in the parameters. The `index` parameter indicates the index of the key returned in the array of stub method `SplitCompositeKey`.

Internally this method calls Hyperledger Fabric's `getStateByPartialCompositeKey`, `splitCompositeKey` and `getState`.

```
func GetByCompositeKey(key string, columns []string, index int)
(interface{}, error)
```

Parameters:

- `key (string)` - Object type provided while creating composite key.
- `column ([]string)` - This is the slice of attributes on which the ledger has to be queried using the composite key.
- `index(int)` - Index of the attribute.

Returns:

- `Interface{}` - Contains the list of assets which are result of this method.
- `error` - Contains any errors if present.

Stub Method

`model.GetNetworkStub`
This method will return the Hyperledger Fabric `chaincodeStub`.

You can get access to the shim stub by calling the `GetNetworkStub` method. This will help you write your own implementation working directly with the assets.

```
func GetNetworkStub() shim.ChaincodeStubInterface
```

Parameters:

- none

Returns:

- `shim.ChaincodeStubInterface` - This is the Hyperledger Fabric chaincode stub.

Other Methods

- `model.GetTransactionId()`
- `model.GetTransactionTimestamp()`
- `model.GetChannelID()`
- `model.GetCreator()`
- `model.GetSignedProposal()`
- `model.GetArgs()`
- `model.GetStringArgs()`
- `model.getId`

`model.GetTransactionId`
Returns the transaction ID for the current chaincode invocation request. The transaction ID uniquely identifies the transaction within the scope of the channel.

```
func GetTransactionId() string
```

Parameters:

- none

Returns:

- `string` - This contains the required transaction ID.

`model.GetTransactionTimestamp`
Returns the timestamp when the transaction was created. This is taken from the transaction `ChannelHeader`, therefore it will indicate the client's timestamp, and will have the same value across all endorsers.

```
func GetTransactionTimestamp() (*timestamp.Timestamp, error)
```

Parameters:

- none

Returns:

- `timestamp.Timestamp` - Contains the timestamp required.
- `error` - Contains any errors if present.

`model.GetChannelID`
Returns the channel ID for the proposal for the chaincode to process.

```
func GetChannelID() string
```

Parameters:

- none

Returns:

- `string` - Contains the required channel ID as a string.

`model.GetCreator`
Returns the identity object of the chaincode invocation's submitter

```
func GetCreator() ([]byte, error)
```

Parameters:

- none

Returns:

- `[ ]byte` - Contains the required identity object serialized.
- `error` - Contains any errors if present.

`model.GetSignedProposal`
Returns a fully decoded object of the signed transaction proposal.

```
func GetSignedProposal() (*peer.SignedProposal, error)
```

Parameters:

- none

Returns:

- `*peer.SignedProposal` - Contains the required signed proposal object.
- `error` - Contains any errors if present.

`model.GetArgs`
Returns the arguments as array of strings from the chaincode invocation request.

```
func GetArgs() [][]byte
```

Parameters:

- none

Returns:

- `[ ][ ]byte` - Contains the arguments passed.

`model.GetStringArgs`
Returns the arguments intended for the chaincode Init and Invoke as a string array.

```
func GetStringArgs() []string
```

Parameters:

- none

Returns:

- `[]string` - Contains the required arguments as a string array.

`model.getId`
When the asset has a derived key as `Id`, you can use this method to get a derived ID. This method will return an error if the derived key contains `%t` (timestamp).

Parameters:

- `object` - Object should contain all the properties on which the derived key is dependent.

Returns:

- Returns the derived key as a string.

Example:

```
func (t *Controller) CustomGetterForSupplier(License string, Name
string)(interface{}, error){
   var asset Supplier
   asset.License = License
   asset.Name = Name
   id,err := model.GetId(&asset)
   if err !=nil {
      return nil, fmt.Errorf("error in getting ID %s", err.Error())
   }
   return t.GetSupplierById(id)
}
```

Utility Package

The following methods in the utility package may be useful:

`Util.CreateModel`
Parses the provided JSON string and creates an asset object of the provided type.

```
func CreateModel(obj interface{}, inputString string) error
```

Parameters:

- `inputString (string)` - The input JSON string from which the object is to be created.
- `obj (interface{})` - The reference of the object that is to be created from the JSON string. This object will store the created model which is also validated as per validator tags.

Returns:

- `error` - Contains any errors found while creating or validating the asset.

`util.ConvertMapToStruct`
Convert the provided map into object of provided type.

```
func ConvertMapToStruct(inputMap map[string](interface{}), resultStruct
interface{}) error
```

Parameters:

- `inputMap (map[string](interface{}))` - Map which needs to be converted into the asset object.
- `resultStruct (interface{})` - The reference of the required asset object which needs to be generated from the map. Contains the result asset object required.

Returns:

- `error` - Contains any errors found while creating or validating the asset.

Controller

The `Controller.go` file implements the CRUD and custom methods for the assets.

You can create any number of classes, functions, or files, but only those methods that are defined on chaincode struct are invokable from outside, the rest of them are hidden.

CRUD Methods

As described in Input Specification File, you can specify which CRUD methods you want generated in the specification file. For example, if you selected to generate all methods, the result would be similar to:

```
//
//Supplier
//
func (t *ChainCode) CreateSupplier(inputString string) (interface{},
error) {
    var obj Supplier
    err := util.CreateModel(&obj, inputString)
    if err != nil {
        return nil, err
    }
    return model.Save(&obj)
}

func (t *ChainCode) GetSupplierById(id string) (interface{}, error) {
    asset, err := model.Get(id)
    return asset, err
}

func (t *ChainCode) UpdateSupplier(inputString string) (interface{}, 
error) {
    var obj Supplier
    err := util.CreateModel(&obj, inputstring)
    if err != nil {
        return nil, err
    }
return model.Update(&obj)
}

func (t *ChainCode) DeleteSupplier(id string) (interface{}, error) {
    return model.Delete(id)
}

func (t *ChainCode) GetSupplierHistoryById(id string) (interface{}, 
error) {
    historyArray, err := model.GetHistoryByld(id)
    return historyArray, err
}

func (t *ChainCode) GetSupplierByRange(startkey string, endKey string) 
(interface{}, error) {
    assetArray, err := model.GetByRange(startkey, endKey)
    return assetArray, err
}
```

Custom Methods

The following custom methods were generated from our example specification file.

The `executeQuery` shows how SQL rich queries can be called. The validators against the arguments are added automatically by Blockchain App Builder based on the type of the argument specified in the specification file.

You can implement the functionality as per the business logic.

```
//
//Custom Methods
//
/*
*    BDB sql rich queries can be executed in OBP CS/EE.
*    This method can be invoked only when connected to remote OBP CS/EE 
network.
*/
```

```
func (t *ChainCode) ExecuteQuery(inputQuery string) (interface{},
error) {
    resultArray, err := model.Query(inputQuery)
    return resultArray, err
}

func (t *ChainCode) FetchRawMaterial(supplierId string,
rawMaterialSupply int) (interface{}, error) {
    return nil, nil
}

func (t *ChainCode) GetRawMaterialFromSupplier(manufacturerId string,
supplierId string, rawMaterialSupply int) (interface{} error) {
    return nil, nil
}

Func (t *ChainCode) CreateProducts(manufacturerId string,
rawMaterialConsumed int, productsCreated int) (interface{}, error) {
    return nil, nil
}

func (t *ChainCode) SendProductsToDistribution() (interface{}, error) {
    return nil, nil
}
```

For Go chaincodes, every custom method should return two values: *empty interface*, *error*. For example:

```
func (t *Controller) FetchRawMaterial(supplierId string,
rawMaterialSupply int) (interface{}, error) {
    return nil, nil
}
```

Init Method

We have provided one `init` method in the controller with an empty definition. This method will be called by the Hyperledger Fabric `Init` method during first time instantiation or upgrade of a chaincode.

```
type Controller struct {
}
func (t *Controller) Init(args string) (interface{}, error)
    { return nil, nil
}
```

If you would like to initialize any application state at this point, you can use this method to do that.

Deploy Your Chaincode Using Visual Studio Code

Once your chaincode project is created, you can deploy it locally to the automatically generated Hyperledger Fabric network, or remotely to your Oracle Blockchain Platform

Cloud or Enterprise Edition. You can also package the chaincode project for manual deployment to Oracle Blockchain Platform.

- Deploy the Chaincode to a Local Hyperledger Fabric Network
- Deploy Your Chaincode to a Remote Oracle Blockchain Platform Network
- Package Your Chaincode Project for Manual Deployment to Oracle Blockchain Platform

Deploy the Chaincode to a Local Hyperledger Fabric Network

Once you have created your chaincode project, you can test it in a local Hyperledger Fabric basic network.

When you install the Blockchain App Builder extension for VS Code, it automatically creates a Hyperledger Fabric network with a single channel. This will be listed as `Local Environment` in the **Environments** pane. You can't delete or modify this environment; you can just deploy chaincodes to it and rebuild it if it stops working correctly.

Blockchain App Builder chaincode deployment starts the Hyperledger Fabric basic network, other services, and installs and instantiates the chaincode for you.

1. In the **Chaincode Details** pane, select **Deploy**.
2. In the deployment wizard:
   - Ensure the correct chaincode name is selected.
   - Select your target environment - for local deployment choose **Local Environment**.
   - Select the channel you want to deploy to. A channel named "`mychannel`" is created by default with the extension's installation, and can be used for testing.
   - Optionally enter any initial parameters that may be required.
3. Click **Deploy**.

When the chaincode has finished instantiating, the **Output** console will state that it has successfully instantiated it on the given channel, installed, and deployed the chaincode.

Troubleshooting

You may encounter the following issues when running your chaincode project on a local network.

Missing Go permissions
While installing Go chaincode project in local network, you might see an error similar to the following in the **Output** console:

```
INFO (Runtime): 2020/06/22 22:57:09 build started

INFO (Runtime): Building ....

INFO (Runtime): go build runtime/cgo: copying /Users/myname/
```

```
Library/Caches/go-build/f8/.…..….d: open /usr/local/go/pkg/darwin_amd64/
runtine/
cgo.a: permission denied

ERROR (Runtime): go build runtine/cgo: copying /Users/myname/Library/
Caches/go-build/f8/.…..….d: open /usr/local/go/pkg/darwin_amd64/runtime/
cgo.a: permission denied

INFO (Runtime): An error occurred while building: exit status 1
```

This is due to missing permissions for Go. This error has been seen only in Mac OS. This is a known issue:

- https://github.com/golang/go/issues/37962
- https://github.com/golang/go/issues/24674
- https://github.com/udhos/update-golang/issues/15

Solution: change the permissions of your `$GOROOT` and try deploying again:

```
sudo chmod -R 777 /usr/local/go
```

Instantiation failure
Due to instantiation failure, corrupt instantiation, a Docker peer container being full, or a Docker peer being killed in the local network, you may see an error similar to:

```
============ Started instantiate Chaincode ============
[2028-19-01T19:25:lO.372] [ERROR] default - Error instantiating
Chaincode GollGl on channel mychannel, detailed
error: Error: error starting container: error starting container:
Failed to generate platform-specific docker
build: Failed to pull hyperledger/fabric-ccenv:latest : API error
(404): manifest for hyperledger/
fabric-ccenv:latest not found: manifest unknown: manifest unknown
[2020-19-01T19:25:10.372] (INFO) default -
============ Finished instantiate Chaincode ============
[2020-19-01119:25:10.372] [ERROR] default - Error: Error instantiating
Chaincode Goll01 on channel mychannel,
detailed error: Error: error starting container: error starting
container: Failed to generate platfom-specific
docker build: Failed to pull hyperledger/fabric-ccenv: latest : API
error (404): manifest for hyperledger/
fabric-ccenv:lalest not found: manifest unknown: manifest unknown
exited: signal: terminated
INFO: exited: signal: terminated

ERROR: Error in Chaincode deployment
```

This is due to a peer container not able to start up properly again.

Solution: Rebuild your runtime by selecting your local environment in the **Environments** pane, right-clicking and selecting **Rebuild Local Environment**. Attempt to deploy again.

Environment Rebuild Required
You may see an error similar to:

```
Starting ca.example.com ...
Starting orderer.example.com ...
Starting orderer.example.com ... error
ERROR: for orderer.example.com
Cannot start service orderer.example.com:
error while creating mount source
path '/host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/network/basic-
network/config': mkdir /host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0: operation not permitted
Starting ca.example.com... error
ERROR: for ca.example.com
Cannot start service ca.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/crypto-config/peerOrganizations/org1.example.com/
ca': mkdir /host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0: operation not permitted
ERROR: for orderer.example.com
Cannot start service orderer.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/config': mkdir /host_mnt/c/Users/opc/.vscode/
extensions/oracle.oracle-blockchain-1.4.0: operation not permitted
ERROR: for ca.example.com
Cannot start service ca.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/crypto-config/peerOrganizations/org1.example.com/
ca': mkdir /host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0: operation not permitted
Encountered errors while bringing up the project.
ERROR: Starting ca.example.com ...
Starting orderer.example.com ...
Starting orderer.example.com ... error
ERROR: for orderer.example.com
Cannot start service orderer.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/config': mkdir /host_mnt/c/Users/opc/.vscode/
extensions/oracle.oracle-blockchain-1.4.0: operation not permitted
Starting ca.example.com ... error
ERROR: for ca.example.com
Cannot start service ca.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/crypto-config/peerOrganizations/org1.example.com/
ca': mkdir /host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0: operation not permitted
ERROR: for orderer.example.com
```

```
Cannot start service orderer.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/config': mkdir /host_mnt/c/Users/opc/.vscode/
extensions/oracle.oracle-blockchain-1.4.0: operation not permitted
ERROR: for ca.example.com
Cannot start service ca.example.com: error while
creating mount source path '/host_mnt/c/Users/opc/.vscode/extensions/
oracle.oracle-blockchain-1.4.0/node_modules/@oracle/ochain-cli/runtime/
network/basic-network/crypto-config/peerOrganizations/org1.example.com/
ca': mkdir /host_mnt/c/Users/opc/.vscode/extensions/oracle.oracle-
blockchain-1.4.0: operation not permitted
Encountered errors while bringing up the project.
ERROR: Error in Chaincode deployment
```

You need to rebuild your local environment. In the App Builder **Environments** pane, right-click your local environment and select **Rebuild Local Environment**.

Deploy Your Chaincode to a Remote Oracle Blockchain Platform Network

Once you've instantiated and tested your chaincode project on a local network to ensure it's working as designed, you can deploy it to Oracle Blockchain Platform.

Download Files Needed for Your Connection Profile

You can download your connection profile from the Oracle Blockchain Platform Cloud or Enterprise Edition instance:

1. Create a folder in your local storage to put the connection profile, and admin keys and certificates.
2. Download your connection profile. In your Oracle Blockchain Platform instance's console, on the **Developer Tools** tab, **Application Development**, and then **Download the development package**.
3. Retrieve your admin credentials from your instance's console. Click **Network** then **Export Admin Credential from your organization**.
   - Copy the admin certificate from the admin credentials to `artifacts/crypto/ordererOrganizations/<instance-name>/signcert/<instance-name>-signcert.pem` and `artifacts/crypto/peerOrganizations/<instance-name>/signcert/<instance-name>-signcert.pem`.
   - Copy the admin key from the admin credentials to `artifacts/crypto/ordererOrganizations/<instance-name>/keystore/<instance-name>-key.pem` and `artifacts/crypto/peerOrganizations/<instance-name>/keystore/<instance-name>-key.pem`.

Note:

Keep in mind that if you're placing your connection profile in your project folder, if you're synching this with a content repository such as Github you might accidentally share your private keys. Ensure that everything private is added to your `.gitignore` file.

Create a Connection to an Oracle Blockchain Platform Instance

You must have a Blockchain Platform instance up and running to perform this step.

1. In the VS Code **Environments** pane, click the **Create Environment** icon.
2. On the **Environments Details** wizard:
   - Enter the name for your remote environment.
   - Enter a description.
   - Browse to the directory where you've stored your connection profile from Oracle Blockchain Platform.
   - Enter a user name and password with permissions to install, instantiate, and invoke chaincode.
3. Click **Test Login**. This logs your profile into the remote network so that you can deploy to it.
4. If you logged in successfully, click **Create** to save the profile.

Deploy Your Chaincode

1. Select the chaincode project you want to deploy in the **Chaincodes** pane.
2. In the **Chaincode Details** pane, select **Deploy**.
3. In the deployment wizard, the name of the chaincode project should be pre-filled.
   - Select your target environment - for remote deployment choose the Oracle Blockchain Platform environment you set up previously.
   - Enter the name of the channel you want to deploy to.
   - Optionally set any required initial parameters.
4. Click **Deploy**.

Once the chaincode has successfully deployed to the remote Oracle Blockchain Platform, theconsole log will show that:

- It has successfully installed the chaincode project.
- It has successfully instantiated the chaincode on each peer and the channel.

Updating the Chaincode Project

The upgrade of the chaincode is handled automatically by Blockchain App Builder. After you have made changes to your chaincode, just Deploy again - this will automatically perform the update for you.

If your update is successful, the log will show

- It has successfully upgraded the chaincode version.

- It has successfully installed the chaincode project.
- It has successfully instantiated the chaincode on each peer and the channel.

Package Your Chaincode Project for Manual Deployment to Oracle Blockchain Platform

You can package your chaincode projects for manual deployment to Oracle Blockchain Platform Cloud or Enterprise Edition.

The **Package** function creates a zip file containing only the build and distribution files - the `binary`, `libs`, `node_modules`, and `test` folders from your chaincode project are not included. This zip can be manually uploaded to Oracle Blockchain Platform for deployment.

1. Select your chaincode project in the **Chaincodes** pane.
2. Right-click and select **Package**.
3. Select a location to save the package, and click **Select Output Folder**.

When the command completes successfully, the location of the package will be returned in the output.

Test Your Chaincode Using Visual Studio Code

If your chaincode is running on a network, you can test any of the generated methods. Additionally, If you chose to create the `executeQuery` method during your chaincode development, you can run SQL rich queries if your chaincode is deployed to an Oracle Blockchain Platform network.

- Test Your Chaincode on a Local Hyperledger Fabric Network
- Testing Lifecycle Operations on a Remote Oracle Blockchain Platform Network
- Execute Berkeley DB SQL Rich Queries

Test Your Chaincode on a Local Hyperledger Fabric Network

Once your chaincode project is running on a local network, you can test it.

Blockchain App Builder contains a built-in wizard to assist you with invoking or querying your chaincode.

1. Select your chaincode project in the **Chaincodes** pane. In the **Chaincode Details** pane, select **Execute**. The chaincode name should already be selected. Ensure the target environment is set to **Local Environment** and the channel will default to the only channel available.
2. In the **Function** field, select your method from the drop-down list. Every method available in the chaincode is listed.
3. In the **Function Param** field, select the **More Actions (...)** button. This will launch a window with available properties for your selected method. Enter the properties, click **Omit** for any non-mandatory property you don't want submitted when you invoke your method, and click **Save**.
4. Click **Invoke**.

The **Output** console window will show that the function has been invoked. Alternatively, in the **Chaincode Actions** pane, the **Function Output** window displays the output. Click the **More Actions (...)** button to see this output formatted.

If you want to save the method and parameters you just ran, you can click **Save** and enter a name and description for it. It will be saved in your chaincode project in the `Queries` folder. To use it again, right-click it and select **Open**.

If you make any changes to the controller file that would alter the methods, select the **Reload** icon at the top of the **Chaincode Execute** pane. The change should now be reflected in the **Function** drop-down list.

Note:

If you don't want to use the wizard for testing, you can also run the Blockchain App Builder command line tools in the Visual Studio Code **Terminal** window. Follow the instructions provided here to test with the command line: Test Your Chaincode on a Local Hyperledger Fabric Network.

Automatic Install and Instantiate After Update

Whenever you update and save your chaincode, the changes will be compiled, installed and instantiated automatically. There is no need to strip down or bring up the local network again. All projects will be automatically compiled and deployed on every change.

Testing Lifecycle Operations on a Remote Oracle Blockchain Platform Network

Once your chaincode project has successfully deployed to your remote Oracle Blockchain Platform network, you can test it as described in Test Your Chaincode on a Local Hyperledger Fabric Network.

You can use the same invoke and query commands to perform all method transactions on a remote Oracle Blockchain Platform Cloud or Enterprise Edition network; everything supported on the local network is also supported on the remote network. Simply select the Oracle Blockchain Platform instance as your target environment when executing your tests.

Execute Berkeley DB SQL Rich Queries

If you chose to create the `executeQuery` method during your chaincode development, you can run SQL rich queries if your chaincode is deployed to an Oracle Blockchain Platform network.

If you have used `executeQuery` in the `customMethods` section of the specification file, a corresponding `executeQuery` method will be created in the controller.

Specification file:

```
customMethods:
    - executeQuery
    - "fetchRawMaterial(supplierid: string, rawMaterialSupply: number)"
    - "getRawMaterialFromSupplier(manufacturerId: string, supplierld: 
string, rawMaterialSupply: number)"
```

```
    - "createProducts(manufacturerId: string, rawMaterialConsumed:
number, productsCreated: number)"
    - "sendProductsToDistribution()"
```

Controller file:

```
**
*
* BDB sql rich queries can be executed in OBP CS/EE.
* This method can be invoked only when connected to remote OBP CS/EE
network.
*
*/
@Validator(yup.string())
public async executeQuery(query: string) {
    const result = await OchainController.query(query);
    return result;
}
```

You can invoke this method to execute Berkeley DB SQL rich queries on Oracle Blockchain Platform network, ensuring that you select the Oracle Blockchain Platform environment that you created as your target environment when running the queries.

Example:

1. In the **Chaincode Details** pane, select **Execute**. The chaincode name, target environment, and channel should already be pre-filled from the deployment step.
2. In the **Function Name** field, select `executeQuery` from the drop-down list.
3. In the **Function Param** field, select the **More Actions (...)** button. This will launch a window where you can enter the query string. Enter the arguments for your query, and click **Save**.
4. Click **Query**.

The **Output** window and the will show the query being executed and the result.

```
> ochain query executeQuery "SELECT key, valueJson FROM <STATE> WHERE
json_extract(valueJson, '$.rawMaterialAvailable') = 4"
```

The entire SQL query is taken in the argument, so you can make changes to your query on the fly.

Synchronize Specification File Changes With Generated Source Code

You can use this function to bring new changes from the specification file to the chaincode source files (model and controller). This function works with both TypeScript and Go projects.

Note:

- Code sync is unidirectional - you can bring changes from your specification file into your chaincode project, but not the other way around. Changes made in your chaincode project remain as-is after the synching process.

- This command only works if the chaincode project has been scaffolded using a specification file. Ensure you do not delete, rename or move the specification file if you plan to sync any changes from the specification file to the source code in future.

To synchronize your specification and chaincode files:

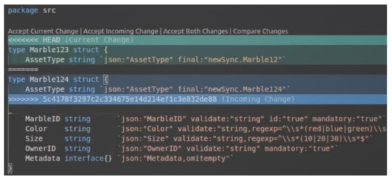
1. In the **Specifications** pane, select the specification file that you've updated to open its **Specification Details** pane. At the top of the pane, click **Chaincodes** to open the pane showing which chaincodes this specification has been used to generate.
2. Select the **Sync** checkbox beside each chaincode you want to update with these changes. You can synchronize more than one chaincode generated from a specification file at a time. Click **Synchronize**.

The chaincode projects should now contain update files.

Resolving Conflicts

Since you can edit both the synchronization files and chaincode files, it's possible to end up with conflicts where the updated specification file would overwrite a change you've made to the chaincode file. In these cases, when you attempt to synchronize an error displays stating there's a conflict. You can use the **Conflicts** pane to resolve these.

1. On the **Conflicts** pane, click the name of the chaincode file where the conflicts exist. The file will open in an editor with the conflicts highlighted.


Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes

```
package src

<<<<<<< HEAD (Current Change)
type Marble123 struct {
    AssetType string `json:"AssetType" final:"newSync.Marble12"`
=======
type Marble124 struct {
    AssetType string `json:"AssetType" final:"newSync.Marble124"`
>>>>>>> 5c4178f3297c2c334675e14d214ef1c3e832de88 (Incoming Change)

    MarbleID string      `json:"MarbleID" validate:"string" id:"true" mandatory:"true"`
    Color    string      `json:"Color" validate:"string,regexp=^\\s*(red|blue|green)\\s
    Size     string      `json:"Size" validate:"string,regexp=^\\s*(10|20|30)\\s*$"`
    OwnerID  string      `json:"OwnerID" validate:"string" mandatory:"true"`
    Metadata interface{} `json:"Metadata,omitempty"`
}
```

   In the example shown, `Marble124` is in the specification file, and `Marble123` is in the chaincode model file.

2. Above the conflict are your list of options. Click **Accept Current Change** to override the specification file and use what is currently in the chaincode file. Click **Accept Incoming Change** to override the chaincode file and use what is currently in the specification file.
3. Return to the **Conflicts** pane. Select the **Sync** checkbox next to the conflict name, and click **Confirm Changes**. If you have multiple conflicts, resolve all of them before before clicking **Confirm Changes**.

Debugging from Visual Studio Code

Blockchain App Builder includes line-by-line debug support from Visual Studio Code for both TypeScript and Go projects.

To run line-by-line debugging:

1. Open your chaincode project in VS Code Explorer. In the **Chaincodes** pane, right-click your chaincode and select **Open in Explorer**.
2. Attach breakpoints to your code wherever necessary.

3. Go to the **Run** menu and click **Start Debugging**. This attaches the debugger. It may take several seconds for the debugger to attach to the chaincode.
4. Call any command from the Terminal which would execute your code.
If you've been using the VS Code interface to test your chaincode so far, you can follow the invocation syntax outlined in Test Your Chaincode on a Local Hyperledger Fabric Network.

   The debugger will stop at your breakpoints. You can then start the debugging.
5. Restart debugging to reflect new changes.

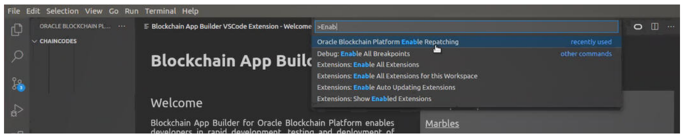
Since the chaincode is running in debug mode, the hot deployment of new changes does not happen automatically. You must manually restart the debugging process, using the debug controls in VS Code, in order to make the latest changes take effect.

Troubleshoot Blockchain App Builder VS Code Extension

The following can be used to troubleshoot system problems with Blockchain App Builder VS Code extension.

Repatching
By default, when you install the extension all the patches are installed for you. Patching is done so that VS Code can connect to Oracle Blockchain Platform. However you may encounter network or other problems that prevent the patching process from completing successfully. In these cases, you can manually force the Blockchain App Builder to repatch itself.

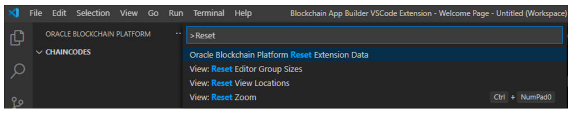
1. Open the Command Palette from the **View** menu.
2. In the Command Palette, type `Enable Repatching`.


File Edit Selection View Go Run Terminal Help
ORACLE BLOCKCHAIN PL...
CHAINCODES
Blockchain App Builder VSCode Extension - Welcome
>Enab
Oracle Blockchain Platform Enable Repatching
recently used
Debug: Enable All Breakpoints
other commands
Extensions: Enable All Extensions
Extensions: Enable All Extensions for this Workspace
Extensions: Enable Auto Updating Extensions
Extensions: Show Enabled Extensions
Blockchain App Builc
Welcome
Blockchain App Builder for Oracle Blockchain Platform enables
Marbles

3. Select **Oracle Blockchain Platform Enable Repatching**. VS Code will clear the existing patching data and force a repatch.

Resetting Extension Data
It is possible for your Blockchain App Builder user data to get corrupted. This option clears your data from Blockchain App Builder without impacting anything stored locally.

1. Open the Command Palette from the **View** menu.
2. In the Command Palette, type `Reset Extension`.


File Edit Selection View Go Run Terminal Help
Blockchain App Builder VSCode Extension - Welcome Page - Untitled (Workspace)
ORACLE BLOCKCHAIN PLATFORM
CHAINCODES
>Reset
Oracle Blockchain Platform Reset Extension Data
View: Reset Editor Group Sizes
View: Reset View Locations
View: Reset Zoom
Ctrl + NumPad0

3. Select **Oracle Blockchain Platform Reset Extension Data**. VS Code will clear the existing blockchain data and reload the default installation data. This will not affect the files stored locally in your system, but you will have to import them back into VS Code and reconfigure any environments you had previously set up.

Migrating Chaincode Between Blockchain App Builder Versions

If you have existing chaincode projects created with earlier versions of Blockchain App Builder, you can migrate them to use the new features of the updated Blockchain App Builder. If you don't need to use any of the updated Blockchain App Builder features, you don't need to migrate your chaincode projects.

To migrate your chaincode project:

1. Create a backup of your existing chaincode project folder. For example, if your chaincode project was in `TestCC`, copy the contents to `TestCC_bak`.
2. Install the updated Blockchain App Builder.
3. Create a new chaincode project with the same name as your existing chaincode project (`TestCC`). If using Go, ensure you're using the same root folder.
4. Copy the `src` folder from the `TestCC_bak` backup, and use it to replace the `src` folder in the newly created project.

D

Run Solidity Smart Contracts with EVM on Oracle Blockchain Platform

This topic provides a walkthrough showing how you can run Solidity smart contracts with EVM (Ethereum Virtual Machine) deployed as a chaincode on Oracle Blockchain Platform.

The Ethereum Virtual Machine runs smart contracts in the Ethereum networks. It was created through the Hyperledger Burrow project and has been integrated into Hyperledger Fabric. This project enables you to use a Hyperledger Fabric permissioned blockchain platform to interact with Ethereum smart contracts written in an EVM-compatible language such as Solidity. See: Hyperledger Fabric EVM chaincode.

A basic overview of the process of running a Solidity smart contract on a provisioned Oracle Blockchain Platform:

1. Upload the EVM chaincode zip into Oracle Blockchain Platform.
2. Deploy it using the bytecode generated by the Remix IDE.
3. Get the smart contract address in response to the deployment, and use that address to send transactions.

Steps in this topic have been tested with `fabric-chaincode-evm:release-0.4` and may not work with other releases.

Set Up the EVM Chaincode Zip File

Before deploying the smart contract, the EVM chaincode zip needs to be prepared. To create the chaincode zip folder:

1. Download the EVM chaincode zip from Hyperledger Fabric EVM chaincode.
2. Unzip the downloaded `fabric-chaincode-evm-release-0.4.zip`.
3. In the extracted files, go to `fabric-chaincode-evm-release-0.4/evmcc/vendor/github.com/hyperledger`.
4. Create the directory `fabric-chaincode-evm/evmcc` in the `/hyperledger` directory.
5. Copy the following folders from `fabric-chaincode-evm-release-0.4/evmcc` to `fabric-chaincode-evm-release-0.4/evmcc/vendor/github.com/hyperledger/fabric-chaincode-evm/evmcc`:
    - `/address`
    - `/event`
    - `/eventmanager`
    - `/mocks`
    - `/statemanager`

6. Zip the top-level `fabric-chaincode-evm-release-0.4/evmcc` folder and rename it. The following steps use `evmcc.zip` as the example name.

Deploy EVM Chaincode on Oracle Blockchain Platform

After you have created the EVM chaincode zip, you need to deploy it on Oracle Blockchain Platform.

1. Log into the Oracle Blockchain Platform console.
2. On the **Chaincodes** tab, click **Deploy a New Chaincode**.
3. Select **Quick Deployment**, and entire the following information:
   - **Chaincode Name:** enter any name of your choice. This example uses `soliditycc`.
   - **Version:** `v0`
   - **Initial Parameters for Chaincode Instantiation:** leave this field empty
   - **Channel:** select the channels on which you want to install and instantiate the chaincode
   - **Chaincode source:** upload the `evmcc.zip` package you created in the previous steps.

   After you submit your information, the EVM chaincode will be visible in the Chaincodes tab.

Create and Compile Your Solidity Smart Contract

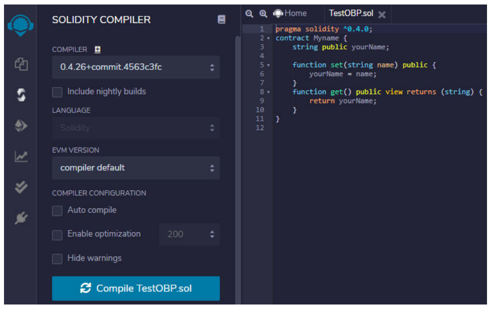
1. Open the browser-based Remix IDE: https://remix.ethereum.org/.
2. If you already have a Solidity smart contract written, import it into Remix.
3. If you don't have a Solidity smart contract written, create a Solidity file (`.sol`) in Remix and do one of the following:
   - If you're familiar with Solidity you can create your own smart contract file.
   - You can use the Simple Storage sample code provided in the Solidity documentation: Solidity: Introduction to Smart Contracts
   - You can use the sample code being used for this example which takes `string name` as an input and prints the same as output string using `set(name)` and `get()`:

   ```
   pragma solidity ^0.4.0;
   contract Myname {
       string public yourName;

       function set(string name) public {
           yourName = name;
       }
       function get() public view returns (string) {
           return yourName;
       }
   }
   ```

   Note that you may see an error message about the default compiler version not matching the version you've specified in your smart contract.

4. Compile your smart contract. Open the Solidity Compiler panel in Remix, ensure that your smart contract tab is open to select it as the file being compiled, set the compiler version to the most recent 4.X version, and click Compile.



COMPILER: 0.4.26+commit.4563c3fc
Include nightly builds
LANGUAGE: Solidity
EVM VERSION: compiler default
COMPILER CONFIGURATION
Auto compile
Enable optimization 200
Hide warnings

```
pragma solidity ^0.4.0;
contract Myname {
    string public yourName;

    function set(string name) public {
        yourName = name;
    }
    function get() public view returns (string) {
        return yourName;
    }
}
```

5. Once the file has compiled, click the **Bytecode** icon - this copies the bytecode as a JSON document to your clipboard.
6. Paste the copied bytecode into a text editor and save it.

Deploy the Smart Contract

In the copied bytecode, the section you need is the `"object"` field. This is the EVM bytecode of a sample smart contract.

```
"object":
"608060405234801561001057600080fd5b506104108061002060003960 00f30060
806040526004361061005757600035 7c0100000000000000000000000000000000000000
0000000000000000000900463
ffffffff1680634ed3885e1461005c5780636d4ce63c146100c5578063d97d6630146101
55575b600080fd5b34801561
006857600080fd5b506100c360048036038101908080359060200190820180359060 2001
908080601f01602080910402
602001604051908101604052809392919081815260200183838082843782019150505050
505091929192905050506101
e5565b005b3480156100d157600080fd5b506100da6101ff565b60405180806020018281
038252838181518152602001
9150805190602001908083836000 5b8381101561011a5780820151818401526020810190
506100ff565b505050509050
90810190601f1680156101475780820380516001836020036101000a031916815260200 1
91505b509250505060405180
910390f35b34801561016157600080fd5b5061016a6102a1565b60405180806020018281
038252838181518152602001
9150805190602001908083836000 5b838110156101aa5780820151818401526020810190
5061018f565b505050509050
90810190601f1680156101d757808203805160018360200361010 00a031916815260200 1
91505b509250505060405180
910390f35b806000908051906020019061 01fb929190610 33f565b5050565b6060600080
546001816001161561010002
0316600290048060 1f01602080910402602001604051908101604052809291908181526 0
2001828054600181600116 15
610100020316600290048015610297578060 1f1061026c576101008083540402835291 60
200191610297565b82019190
6000526020600020905b8154815290600101906020018083116102 7a57829003601f1682
01915b505050505050905090 56
```

```
5b600080546001816001161561010002031660029004806 01f016020809104026020016 0
4051908101604052809291 90
818152602001828054600181600116156101000203166002900480156103375780601f10
61030c576101008083540402
835291602001916103375 65b820191906000526020600020905b815481529060010190 60
200180831161031a57829003
601f168201915b5050505050 81565b828054600181600116156101000203166002900490
600052602060002090601f01
602090048101928260 1f106103805780516 0ff19168380011785556103ae565b82800160
01018555821561 03ae579182
015b828111156103ad57825182559160200191906001019061039256 5b5b5090506103bb
91906103bf565b5090565b61
03e191905b808211156103dd57600081600090555060010161 03c5565b5090565b905600
a165627a7a72305820a990d4
0b57c66329a32a18e847b3c18d6c911487ffadfed2098e71e8cafa0c980029",
```

In general, the EVM expects two arguments:

- The `to` address.
- The `input` that's necessary in Ethereum transactions.

To deploy smart contracts, the `to` field is the zero address, and the `input` is the compiled EVM bytecode of the contract. Thus, there are two arguments provided to the `invoke`. The first one, which was traditionally supposed to be a function name inside the chaincode, is now `0000000000000000000000000000000000000000`, and the second argument is the Solidity smart contract bytecode.

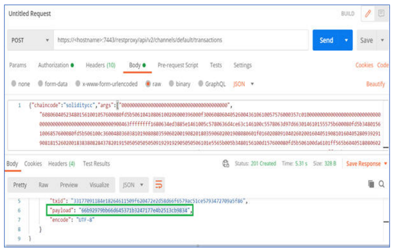
1. To deploy the Solidity smart contract on Oracle Blockchain Platform, you need to invoke the deployed EVM chaincode with these two arguments using REST proxy endpoints.

   ```
   --data-raw '{"chaincode":"<chaincodename>","args":
   ["<zeroaddress>","<EVMbytecode>"]}'
   ```

   For example, using cURL to deploy the Solidity smart contract to Oracle Blockchain Platform with the name `soliditycc`:

   ```
   curl -L -X POST 'https://<hostname>:7443/restproxy/api/v2/channels/
   <channelname>/transactions' \
   -H 'Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=' \
   -H 'Content-Type: application/json' \
   --data-raw '{"chaincode":"soliditycc","args":
   ["0000000000000000000000000000000000000000","608060405234801561....1
   52d2820d3b5d784b3711119691d0029"],"timeout":0,"sync":true}'
   ```

2. The response `payload` of the transaction is the contract address for your deployed contract. Copy this address and save it - it's used when you execute smart contract functions.



Untitled Request

POST https://<hostname>:7443/restproxy/api/v2/channels/default/transactions

```
{"chaincode":"soliditycc","args":["0000000000000000000000000000000000000000",
    "6080604052348015610010576000080fd5b50610410806100206000396000f3006080604052600436106100575760000357c010000000000000000000000000000
    0000000000000000000000000000900463ffffffff1680634ed3885e1461005c5780636d4ce63c146100c5578063d97d663014610155575b600080fd5b3480156
    1006857600080fd5b506100c36004803603810190808035906020019082018035906020019080806010f01602080910402602001604051908101604052809392919
    90818152602001838380828437820191505050505050919291929050505061010e5565b005b3480156100d157600080fd5b506100da6101ff565b60405180806002
```

Status: 201 Created Time: 5.31 s Size: 328 B

```
        "txid": "33177091184e18264611509f620472e2d58d66f6579ac51ce5793472709a5f86",
        "payload": "66b92979bb66d645371b3247177e4b2513cb9834",
        "encode": "UTF-8"
    }
}
```

In this example, the smart contract address is `66b92979bb66d645371b3247177e4b2513cb9834`.

Interacting With the Smart Contract

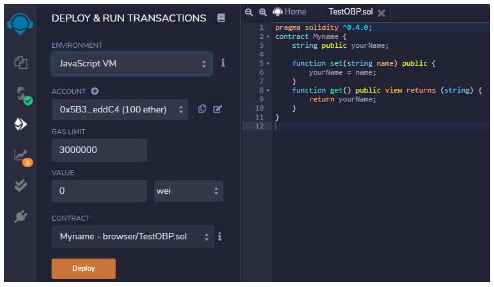
To interact with the deployed smart contract, you need the contract address returned as the payload while deploying the contract in the previous section.

To execute functions, you will use invoke and query transactions but with different parameters. The sample contract contains two functions: `get` and `set`.

In these transactions, the `to` field is the contract address and the `input` field is the function execution hash concatenated with any of the required arguments.

You need to acquire the hash of the function execution to run a transaction. A simple way to do this is to execute the functions in the Remix IDE and to then copy the hash from the transaction logs:

1. In the Remix IDE, open the **Deploy and Run Transactions** panel. Ensure that your contract is selected in the **Contract** field, and click **Deploy**.

   DEPLOY & RUN TRANSACTIONS

   ENVIRONMENT: JavaScript VM

   ACCOUNT: 0x5B3...eddC4 (100 ether)

   GAS LIMIT: 3000000

   VALUE: 0 wei

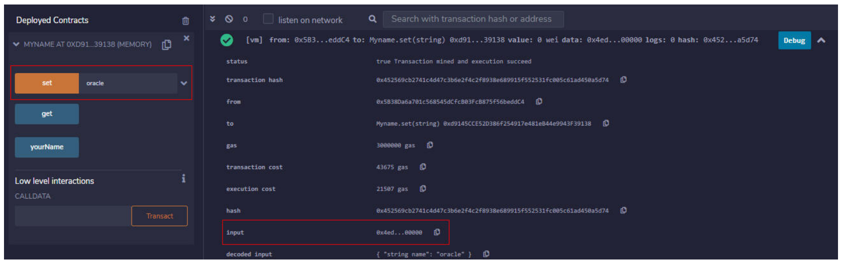
   CONTRACT: Myname - browser/TestOBP.sol

   ```
   pragma solidity ^0.4.0;
   contract Myname {
       string public yourName;

       function set(string name) public {
           yourName = name;
       }
       function get() public view returns (string) {
           return yourName;
       }
   }
   ```

   Once the deployment completes, the contract should be listed in the **Deployed Contracts** list.

2. Expand the contract in the **Deployed Contracts** list. The smart contract functions are listed.
3. Run a transaction. For the provided example, enter `oracle` and click **set**.
4. The Terminal window shows the transaction logs. If the transaction logs are minimized, expand them by clicking the log. Copy the value of the `input` field (which is the function execution hash) by clicking the icon next to it. Save this value to the same location as your contract address, removing the leading `0x`.



```
[vm] from: 0x5B3...eddC4 to: Myname.set(string) 0xd91...39138 value: 0 wei data: 0x4ed...00000 logs: 0 hash: 0x452...a5d74
status              true Transaction mined and execution succeed
transaction hash    0x452569cb2741c4d47c3b6e2f4c2f8938e689915f552531fc005c61ad450a5d74
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  Myname.set(string) 0xd9145CCE52D386f254917e481eB44e9943F39138
gas                 3000000 gas
transaction cost    43675 gas
execution cost      21507 gas
hash                0x452569cb2741c4d47c3b6e2f4c2f8938e689915f552531fc005c61ad450a5d74
input               0x4ed...00000
decoded input       { "string name": "oracle" }
```

5. Once you have the function execution hash and the contract address, you can run the set transaction on Oracle Blockchain Platform using them as the raw data arguments.

   ```
   --data-raw '{"chaincode":"<chaincodename>","args":
   ["<contractaddress>","<setfunctionexecutionhash>"]}'
   ```

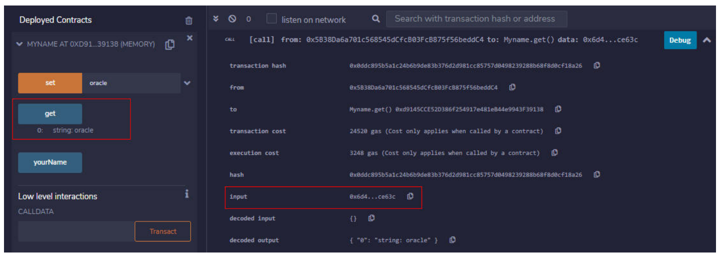
   For example using cURL:

   ```
   curl -L -X POST 'https://<hostname>:7443/restproxy/api/v2/channels/
   <channelname>/transactions' \
   -H 'Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=' \
   -H 'Content-Type: application/json' \
   --data-raw '{"chaincode":"soliditycc","args":
   ["66b92979bb66d645371b3247177e4b2513cb9834","4ed3885e0000000000000000
   0000000000000000000000000000000000000000000200000000000000000000
   000000000000000000000000000000000000000066f7261636c6500000000000000
   0000000000000000000000000000000000000000"]}'
   ```

6. Open the Oracle Blockchain Platform console; the transaction should be listed in the ledger.

To run another transaction such as a query using the smart contract's `get` function, you can generate the function execution hash in Remix and combine it with the contract address:

1. In Remix on the **Deploy and Run Transactions** panel, ensure that your contract is still listed under **Deployed Contracts**. If not, redeploy it.
2. Click **get**. Retrieve and save the input from the transaction as you did with the **set** transaction, removing the leading `0x`.



```
[call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: Myname.get() data: 0x6d4...ce63c
transaction hash    0x0ddc895b5a1c24b6b9de83b376d2d981cc85757d0498239288b68f8d0cf18a26
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  Myname.get() 0xd9145CCE52D386f254917e481eB44e9943F39138
transaction cost    24520 gas (Cost only applies when called by a contract)
execution cost      3248 gas (Cost only applies when called by a contract)
hash                0x0ddc895b5a1c24b6b9de83b376d2d981cc85757d0498239288b68f8d0cf18a26
input               0x6d4...ce63c
decoded input       {}
decoded output      { "0": "string: oracle" }
```

3. You can use this transaction hash and the contract address to run a query transaction against the chaincode deployed on Oracle Blockchain Platform.

```
--data-raw '{"chaincode":"<chaincodename>","args":
["<contractaddress>","<getfunctionexecutionhash>"]}'
```

   For example in cURL:

```
curl -L -X POST 'https://<hostname>:7443/restproxy/api/v2/channels/
<channelname>/chaincode-queries' \
-H 'Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=' \
-H 'Content-Type: application/json' \
--data-raw '{"chaincode":"soliditycc","args":
["66b92979bb66d645371b3247177e4b2513cb9834","6d4ce63c"]}'
```

   The returned payload will contain the asset being queried - in the example case the string `oracle`.